A Flexible System for the Launch and Reconstruction of Localized Acoustic Waves

By

Isaac Jing Herng Leung

B.A.Sc. (Engineering Physics) University of British Columbia

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Applied Science in
The Faculty of Graduate Studies Electrical Engineering

We accept this thesis as conforming to the required standard

The University of British Columbia

May 1996

© Isaac J. H. Leung, 1996
In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Electrical Engineering
The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date: June 7, 1966
Abstract

A synthetic array system for launching and reconstructing localized acoustic waves has been developed. This low cost system is easily changed to many different configurations and for many different experiments. The major components of this system consist of an ultrasonic transducer transmitter-receiver pair, a high bandwidth amplifier, a single axis motion stage and PC-based data generation, acquisition and control cards.

In addition to the design and assembly of this system, the equipment was further characterized so that the performance characteristics can be accounted for in future experiments.

The signal amplitude decay with increasing transmitter-receiver distance behaves approximately as predicted by theory. The amplitude remains relatively constant up to the near field boundary. In the far field, the signal amplitude decays at a rate inversely proportional to the transducer separation.

Transducer response as a function of the angle between the transmitter and receiver studied. Experimental measurements showed that transmitted fields are highly directional, similar to that predicted by theory.

A method was devised to reconstruct received signals. Signals are first processed to remove unwanted low frequency noise. The signal is then integrated twice to recover the original signal, as predicted by theory.

As a final confirmation of the operation of this system, a sample FWM pulse was generated using a $21 \times 21$ array of sources at a distance of 12 cm from the source array. The
reconstructed result is compared with the theoretically reconstructed result as well as the ideal FWM pulse. The experimental results show characteristics similar to that which is predicted by theory.

All required equipment and procedures have been devised to launch localized acoustic waves with this system. The experimental setup is ready to begin further experiments in localized acoustic waves.
# Table of Contents

ABSTRACT.......................................................................................................................... i

TABLE OF CONTENTS........................................................................................................ iii

LIST OF FIGURES............................................................................................................... viii

ACKNOWLEDGEMENTS..................................................................................................... xi

BIBLIOGRAPHY.................................................................................................................. 71

APPENDICES..................................................................................................................... 74

1. INTRODUCTION........................................................................................................... 1

2. BACKGROUND............................................................................................................... 5

   2.1. LOCALIZED WAVES............................................................................................. 5

   2.2. GENERATING LOCALIZED WAVES ..................................................................... 8

   2.3. PULSE RECONSTRUCTION................................................................................... 9

3. EXPERIMENTAL SETUP............................................................................................... 14

   3.1. PROJECT GOALS................................................................................................. 14

      3.1.1. Specifications:............................................................................................. 14

   3.2. SYSTEM DESIGN................................................................................................. 14

   3.3. EXPERIMENTAL EQUIPMENT:.......................................................................... 15
### 3.4. EXPERIMENTAL SETUP

- 3.4.1.1. Gage CompuGen 840A ................................................................. 17
- 3.4.1.2. Gage CompuScope Lite ............................................................. 18
- 3.4.1.3. Keithley-Metrabyte MSTEP 3 ..................................................... 19
- 3.4.1.4. API CMD-50 Stepper Motor Driver ........................................... 20
- 3.4.1.5. DCI 12” Motion Stage and Water Tank ....................................... 22
- 3.4.1.6. Panametrics 3.5MHz Ultrasonic Transducers ............................ 25
  - 3.4.1.6.1. Initial preparation of the transducers for use .......................... 27
  - 3.4.1.6.2. Signal Generation and Reception Accuracy ............................ 27
  - 3.4.1.6.3. Driving the ultrasonic transducer .......................................... 30

### 4. EXPERIMENTAL RESULTS

- 4.1. AMPLITUDE DECAY WITH DISTANCE ............................................. 36
  - 4.1.1. Background .................................................................................. 36
  - 4.1.2. Measurements ............................................................................. 38
    - 4.1.2.1. Decay at 3.5MHz .................................................................. 39
    - 4.1.2.2. Decay at 2.5 MHz ................................................................. 41
    - 4.1.2.3. Decay of FWM Source Pulse .................................................. 43
- 4.2. AMPLITUDE DECAY WITH ANGLE ............................................... 45
  - 4.2.1. Background ................................................................................ 45
  - 4.2.2. Measurements ............................................................................. 47
    - 4.2.3. R = 3 cm ................................................................................. 48
    - 4.2.4. R = 12 cm .............................................................................. 49
# Table of Contents

4.2.5. $R = 20 \text{ cm}$ ......................................................... 50

4.3. SIGNAL RECONSTRUCTION .................................................. 51

4.3.1. Reconstruction by integration .......................................... 52

4.3.1.1. Background ............................................................ 52

4.3.1.2. Measurements .......................................................... 53

4.3.1.3. Curve Fitting the Offset ........................................... 57

4.3.1.4. High Pass Filtering to Remove Offset ............................. 58

4.4. RECONSTRUCTION FROM SYNTHETIC ARRAY ....................... 60

5. CONCLUSIONS AND RECOMMENDATIONS ................................ 66

5.1. RESULTS .......................................................................... 66

5.2. RECOMMENDATIONS .......................................................... 68

APPENDIX A: EXPERIMENTAL EQUIPMENT ................................. 75

A.1 Signal Amplifier ............................................................... 75

A.1.1 Amplifier Design ............................................................ 75

A.1.2 Amplifier Performance Characteristics ............................... 77

APPENDIX B: PROGRAM FILES .................................................. 81

B.1 C Program Files ............................................................... 81

840_ctrl.h ........................................................................... 81

840_ctrl.c ........................................................................... 82

control.c ............................................................................. 92

cs_ctrl.h ............................................................................ 94

cs_ctrl.c ............................................................................ 94
<table>
<thead>
<tr>
<th>Table of Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>delays.h</td>
<td>98</td>
</tr>
<tr>
<td>delays.c</td>
<td>98</td>
</tr>
<tr>
<td>expmt.h</td>
<td>101</td>
</tr>
<tr>
<td>expmt.c</td>
<td>101</td>
</tr>
<tr>
<td>ms3_ctrl.h</td>
<td>111</td>
</tr>
<tr>
<td>ms3_ctrl.c</td>
<td>111</td>
</tr>
<tr>
<td>m_create.c</td>
<td>113</td>
</tr>
<tr>
<td>B.1.2 Maple Program Files</td>
<td>116</td>
</tr>
<tr>
<td>draw_fwm.m</td>
<td>118</td>
</tr>
<tr>
<td>fld_calc.m</td>
<td>119</td>
</tr>
<tr>
<td>f_source.m</td>
<td>120</td>
</tr>
<tr>
<td>fwm_src.m</td>
<td>120</td>
</tr>
<tr>
<td>fwmrecon.m</td>
<td>122</td>
</tr>
<tr>
<td>process.m</td>
<td>123</td>
</tr>
<tr>
<td>processa.m</td>
<td>125</td>
</tr>
<tr>
<td>processf.m</td>
<td>127</td>
</tr>
<tr>
<td>psi_fwm.m</td>
<td>129</td>
</tr>
<tr>
<td>response.m</td>
<td>129</td>
</tr>
<tr>
<td>B.1.3 MatLab Program Files</td>
<td>118</td>
</tr>
<tr>
<td>make.bat</td>
<td>132</td>
</tr>
<tr>
<td>motion.cmd</td>
<td>132</td>
</tr>
<tr>
<td>move6.bas</td>
<td>132</td>
</tr>
<tr>
<td>B.1.4 QuickBASIC Program Files</td>
<td>132</td>
</tr>
</tbody>
</table>
List of Figures

Figure 2-1: FWM pulse at distance 0.0357 m out from origin. Parameters $k=88$ m$^{-1}$ and $z_0=6 \times 10^{-4}$ m. 7

Figure 2-2 FWM source pulse for $r=0$. 9

Figure 2-3 FWM source at $r = 9$mm 9

Figure 2-4 Theoretical FWM pulse. $k=9$ m$^{-1}$, $z_0=6E-5$ m. 11

Figure 2-5 Reconstructed FWM. $k=9$ m$^{-1}$, $z_0=6E-5$ m, 21x21 elements, element spacing=0.92mm 12

Figure 3-1 Overview of Experimental Setup 16

Figure 3-2 CG840A I/O ports, viewed from rear of PC. 17

Figure 3-3 CSLITE I/O Ports, viewed from rear of PC. 18

Figure 3-4 MSTEP-3 Connector, viewed from rear of PC. 19

Figure 3-5 CMD-50 Stepper Motor Driver and required connections. 20

Figure 3-6 CMD-50 Stepper Motor Driver with installed heat sink. 21

Figure 3-7 Connections to control translation stage. 22

Figure 3-8 Motion Stage setup and Time-of-Flight estimates. 24

Figure 3-9 3.5MHz Ultrasonic Transducer 25

Figure 3-10 Schematic of Experiment setup. 27

Figure 3-11 2MHz bandwidth sinc(t) signal. 28

Figure 3-12 8MHz bandwidth sinc(t) signal. 29

Figure 3-13 FWM source pulse at $r = 0$ mm. 30

Figure 3-14 FWM source pulse at $r = 9$ mm. 30

Figure 3-15 2MHz bandwidth signal with transducer. 31

Figure 3-16 8MHz bandwidth signal with transducer. 33

Figure 3-17 FWM source at $r = 0$ cm with transducer. 34
List of Figures

Figure 3-18 FWM source at r = 1.8 cm with transducer. 34
Figure 3-19 FWM source at r = 0 cm, x10 gain. 35
Figure 3-20 FWM source at r = 1.8 cm, x10 gain. 35
Figure 4-1 Plot of approximate Near Field Length vs. Frequency for 6mm source. 37
Figure 4-2 Single cycle of 3.5 MHz sine. 40
Figure 4-3 Received signal characteristics. 40
Figure 4-4 Normalized Amplitude Decay vs. Distance 41
Figure 4-5 Single cycle of 2 MHz sine. 42
Figure 4-6 Received signal characteristics. 42
Figure 4-7 Amplitude Decay with Distance for -2.5 MHz signal. 43
Figure 4-8 Transmitted signal characteristics 44
Figure 4-9 Received signal characteristics 44
Figure 4-10 Amplitude Decay with Distance for FWM source signal 45
Figure 4-11 Beam divergence 46
Figure 4-12 Measuring Amplitude Dependence on Angle. 48
Figure 4-13 Amplitude as a function of the angle between source and receiver. R=3cm. 49
Figure 4-14 Amplitude response with angle at source-receiver distance of R = 12 cm. 50
Figure 4-15 Amplitude response vs. Source-receiver angle at R=20cm. 51
Figure 4-16 FWM source at [8,0]. 54
Figure 4-17 Received signal after integration. 55
Figure 4-18 Integration of received signal + offset. 56
Figure 4-19 Received signal after offset correction and integration. 57
Figure 4-20 FWM source at [0,0]. 58
List of Figures

Figure 4-21 FWM source at [10,0]. 58

Figure 4-22 [8,0] FWM source pulse processed with high pass filter. 59

Figure 4-23 [0,0] FWM source. High pass filter+integration. 60

Figure 4-24 [10,0] FWM source. High pass filter+integration. 60

Figure 4-25 Ideal, theoretical FWM pulse 61

Figure 4-26 Theoretical reconstructed FWM pulse. 21 x 21 sources. 62

Figure 4-27 Experimentally reconstructed pulse. Curve fit correction. 63

Figure 4-28 Experimentally reconstructed pulse. High pass filtering. 63

Figure 4-29 Theoretically reconstructed pulse. Greyscale coded amplitude plot. (Top view of pulse) 64

Figure 4-30 Experimental result (curve fit correction) Greyscale coded amplitude plot. [Top View]. 65

Figure 4-31 Experimental result (High pass filtering). Greyscale-coded amplitude plot. [Top view]. 65

Figure A-1 Amplifier Stage 1 75

Figure A-2 Amplifier Stage 2 76

Figure A-3 Amplifier Stage 3 76

Figure A-4 Amplifier DIN connector pin-outs. 77

Figure A-5 Voltage Divider for limiting the input signal amplitude during testing 78

Figure A-6 Input and Output Signals 78

Figure A-7 Amplifier Frequency Response 79
Acknowledgements

Successful completion of this project would not be possible without the support and direction of my thesis supervisor, Dr. Matthew R. Palmer. His contributions have been invaluable in aiding my completion in a timely manner. I would also like to thank Nelson Ho for his assistance in solving many experimental problems. Finally, I would like to acknowledge those people who believed in me even when I didn’t. Oh, and last of all, let’s give the Great Canadian Beaver three cheers.
Chapter 1: Introduction

1. Introduction

Over the past decade, ultrasonic imaging has become an indispensable tool in many areas of industry. The tools and methods associated with ultrasound imaging are relatively inexpensive, high-speed, robust and simple to operate. In addition, ultrasound is not an ionizing radiation and is therefore considered safe. The item being imaged is not altered or destroyed. These characteristics make it ideal for the two largest fields of application, non-destructive testing and medical imaging. In the general public, ultrasonic imaging is perhaps most well known from the applications in medicine. The ability to produce high resolution, real-time images with no harmful side effects makes this an extremely desirable tool in medicine. In addition to prenatal imaging, which is the most familiar application, ultrasound is also used for cardiac imaging, imaging of abdominal organs, detection of foreign masses in the eye and cranial imaging for specific head trauma cases. More recent developments include the use of Doppler ultrasound for measuring fluid or tissue velocity. Using this technique, cardiac motion or blood flow characteristics can be measured.

While other medical imaging modalities (i.e. X-Ray, CT, PET, etc...) generally make use of transmitted radiation, this is not the case with ultrasonic imaging. In general, the attenuation of sound waves in tissue is far too great for any useful signals to be detected through transmission. While there is active research in transmission ultrasound imaging, few practical applications have yet to be developed.

Most commonly, ultrasound makes use of reflection to construct imagery. As with electromagnetic radiation, reflection occurs at an interface between two media. The strength
of the reflected wave depends on the difference in acoustic impedance between the two media. The acoustic impedance of various soft body tissues does not vary much. This means that only a small portion of the signal is typically reflected back for detection. In addition, secondary effects such as diffraction, absorption and scattering serve to further degrade the reflected signal, limiting the visual quality and resolution of the image.

There have been attempts to overcome some of these problems, especially diffraction, which can be a large contributor to poor resolution. The spreading of the ultrasonic wave through diffraction can be partially overcome by focusing, either through mechanical or electronic means. However, this can bring about other problems related to a specific depth of focus. For example, one method of mechanical focusing makes use of acoustic lenses. The depth of sharpest focus is fixed for a specific lens. In order image at a deeper or shallow depth, a different probe, with a different lens, must be utilized.

While ultrasound remains a valuable tool for qualitative diagnostics, the problems mentioned above limit its usefulness as a quantitative diagnostic tool. Much of the current research in the field of ultrasonic imaging is aimed at rectifying this problem. Approaches include making use of transmission ultrasound, computed tomography (CT) ultrasound and ultrasonic holography.

One of the most successful applications to date has been in the detection of breast cancer, which uses transmission ultrasound. The breast is a fairly ideal organ for quantitative ultrasound. It is relatively limited in thickness and it is composed of mostly uniform tissue. In addition, ultrasonic methods would be preferable to the conventional x-ray mammograms as
ultrasound is so far known to have no harmful side effects, unlike, x-rays, which pose a small, but quantifiable, degree of risk.

Ideally, in order to produce useful, quantitative results with ultrasound, the wave should have the following characteristics: uni-directional, uniform intensity, narrow focus and an extended near field.

Over the past decade, new solutions to the scalar wave equation have been found which predict the possibility of highly localized waves which possess these ideal characteristics. There are many variations of localized waves, but all are related to the Brittingham (1983) waves which have been termed focused wave modes (FWM). These FWM’s, in theory, propagate without diffraction.

FWM’s, just like monochromatic plane waves, have infinite energy and therefore have only theoretical implications. Superpositions or finite-energy approximations, termed localized waves (LW), are physically realizable. The main technical hurdle stems from the strong coupling between spatial and temporal wave components. Each source in an array designed to launch localized waves is required to be independently addressable as each excitation signal is unique.

The goal of this project is to design, construct and to characterize a system for generating acoustic localized waves in water. The system is required to be flexible and easily modified in order to explore many different array configurations. Recent research in our laboratory suggests that a cylindrical source geometry may hold some advantages over simpler planar arrays (Allexandre, 1996).
The system developed through this work is expected to be critical in the further development and refinement of LW imaging systems. Applications include source design, propagation studies, scattering experiments and synthetic aperture imaging.
2. Background

2.1. Localized Waves

Brittingham (Brittingham, 1983) was the first to described localized waves, his version of which has been termed Focused Wave Modes (FWM). He was originally attempting to find solutions to the homogeneous Maxwell's equations with three-dimensional pulse structures which were continuous, non-singular and non-dispersive. The electromagnetic pulses resulting from these solutions carried infinite energy, which originally created some controversy.

This work spurred interest in finding other packet-like solutions. This resulted in many related solutions, including: the modified power spectrum pulse (Ziolkowski, 1985; Ziolkowski, 1989) and X Waves (Lu and Greenleaf, 1992a; Lu et al., 1993).

Palmer and Donnelly (Palmer and Donnelly, 1993) have described the FWM as a solution of the free space scalar wave equation in moving characteristic coordinates. This result is summarized as follows.

The homogenous scalar wave equation is:

\[ (\nabla^2 - \frac{1}{c^2} \frac{\partial^2}{\partial t^2})\psi_h = 0 \]  \hspace{1cm} (2-1)

where: \( \psi_h \) is the scalar wave propagating in a support medium with speed \( c \).

We can arbitrarily choose the \( z \) axis as the axis of propagation and define two characteristic variables \( u = z - ct \) and \( v = z + ct \). Substituting this into Equation (2-1) gives:
Chapter 2: Background

\[(\nabla^2 + 4 \frac{\partial^2}{\partial u \partial v}) \psi_h = 0 \quad (2-2)\]

where: \[\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, \text{ the transverse Laplacian.}\]

One possible set of solutions to Equation (2-2) is given by:

\[\psi_h = \frac{1}{u - u_0} f(v + \frac{\rho^2}{u - u_0}) \quad (2-3)\]

where: \(f\) is a continuous function, with continuous first derivative, \(\rho\) is the transverse distance, \(\rho = \sqrt{x^2 + y^2}\), and \(u_0\) is an arbitrary constant.

Suppose we choose \(u_0 = iz_0\) and \(f(\xi) = e^{ik\xi}\), where \(z_0, k\) are real constants which can be arbitrarily chosen. Substituting into Equation (2-3) gives:

\[\psi_h = \frac{1}{u + iz_0} e^{ik\rho^2} e^{u - u_0} \quad (2-4)\]

This solution is related to Ziolkowski’s original derivation of the FWM solution (denoted here by \(\phi_h\)) by the simple relation:

\[\phi_h = -\frac{1}{4\pi} \psi_h \quad (2-5)\]

If we substitute into Equation (2-5) \(\psi_h\) and our characteristic variables \(u = z - ct\) and \(v = z + ct\), we obtain:
\[ \varphi_n(z, \rho, t) = \frac{1}{4\pi|z_0 + i(z - ct)|} e^{i(k(z + ct))} e^{-\frac{\rho^2}{2}} \] (2-6)

In order for \( \varphi_n \) to be bounded as \( \rho \to \infty \), it is required that \( \frac{k}{z_0} > 0 \). However, as \( k \) and \( z_0 \) can be arbitrarily chosen, this constraint does not present a problem. Varying \( k \) and \( z_0 \) will vary different characteristics of the pulse, including shape and temporal spectrum. Figure 2-1 is a plot of a FWM pulse using the parameter values \( z_0 = 8 \times 10^{-4} \text{ m} \) and \( k = 88 \text{ m}^{-1} \).

![Theoretical FWM](image)

Figure 2-1: FWM pulse at distance 0.0357 m out from origin. Parameters \( k = 88 \text{ m}^{-1} \) and \( z_0 = 6 \times 10^{-4} \text{ m} \).
Chapter 2: Background

2.2. Launching Localized Waves

In this project, a rectangular grid of source elements will be employed in launching acoustic waves. The procedure for designing the excitation depends on an approximate Huygen’s reconstruction. This is derived in Ziolkowski(1991) and also Palmer (1995) and won’t be repeated here. The result is:

\[ s_{FWM} (z, \rho, t) = \frac{\partial \phi_h}{\partial \mu} + \frac{\partial \phi_p}{\partial v} \]  \hspace{1cm} (2-7)

where: \( s_{FWM} \) is the sound pressure distribution on a plane orthogonal to the z-axis.

We can further simplify Equation (2-7) to suit our needs if we make the arbitrary choices that the z-axis will be the chosen axis of propagation and that the pulse starts at \( z=0 \). Including the above assumptions and making the appropriate substitutions for \( u \) and \( v \), we obtain another description for the source functions:

\[ s_{FWM_v} (x, y, z, t) = \phi_h (0, \sqrt{x^2 + y^2}, t) \left( \frac{ik(x^2 + y^2)}{z_0 + i(z - ct)} \right) - \frac{i}{z_0 + i(z - ct)} + ik \]  \hspace{1cm} (2-8)

So, in practice, we merely need to be able to describe the location of a particular source in \( (x,y,z) \) coordinates. Once this is done, Equation (2-8) can be used to calculate the required pulse.

These source functions have two particular characteristics which make them difficult to deal with in practice. One characteristic is that virtually each source function is unique. Only sources that are situated at exactly the same distance from the origin share the same function. Other sources are completely different and are not related by any simple relation.
Chapter 2: Background

The other characteristic is that the sources are broadband functions. In practice, developing equipment that will perform well over a large range of frequencies can be difficult.

![FWM Source at r = 0 mm](image1)

![FWM Source Frequency Spectrum](image2)

Figure 2-2 FWM source pulse for r=0.

![FWM Source at r = 9 mm](image3)

![FWM Source Frequency Spectrum](image4)

Figure 2-3 FWM source at r = 9mm

Figure 2-2 and Figure 2-3 shows two sample source signals at different distances from the origin. As can be seen on the plot of the signal frequency spectrum, the bandwidth of these signals are in the order of MHz.

2.3. Pulse Reconstruction

The final step in the process is to be able to reconstruct an approximation of a localized wave given the known sources. Given an infinite and continuous source array, this
approximation should be exactly equal to $\phi_h$. For a planar array, it can be shown that the
reconstructed signal can be described as a function of the source by:

$$f_R(x, y, z, t) = -\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s_{FWM_m}(x', y', z', t - \frac{R}{c}) \frac{1}{4\pi R} dx' dy'$$  (2-9)

where:

- $R$ is the distance of the observation point to the source location.
- $R = \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2}$ in this case
- $z' = 0$ for a planar array on the $x$-$y$ plane

For the infinite case, $X \to \infty$ and $Y \to \infty$. But of course for the real world, this is not
possible and so $X$ and $Y$ will assume finite values. Furthermore, it is not possible to obtain a
continuous source array, therefore, Equation (2-9) needs to be adapted for the discrete case.

For an array of $(2M+1) \times (2N+1)$ sources, the integration should be expressed as a
summation and the appropriate equation becomes:

$$f_R(x, y, z, t) = -\sum_{n=-N}^{N} \sum_{m=-M}^{M} \frac{1}{4\pi R_{mn}} s_{FWM_m}(m\Delta x, n\Delta y, z', t - \frac{R_{mn}}{c}) \Delta x \Delta y$$  (2-10)

where:

- $\Delta x$ is the element spacing in the $x$ direction
- $\Delta y$ is the element spacing in the $y$ direction
- $R_{mn}$ is the distance from the observation point to the $[m,n]$ array element
- $R_{mn} = \sqrt{(x-m\Delta x)^2 + (y-n\Delta y)^2 + (z-z')^2}$ in this case
- $z' = 0$ for a planar array on the $x$-$y$ plane
Using Equation (2-10), we can now fully simulate the results of launching a FWM pulse from an $M \times N$ array of discrete sources at any distance out from the planar source along the $z$-axis.

Figure 2-4 shows the theoretical FWM pulse at some distance out from the source (12 cm). Figure 2-5 shows a reconstructed FWM pulse, at the same distance and using the same parameters. The simulated array has $21 \times 21$ source elements and spacing between the elements is 0.92 mm.
For computational purposes, we can express Equation (2-10) in a different way. If we assume a rectangular array centered at the origin with equally spaced sources in both the horizontal and vertical directions. Then, we can make use of the symmetry of the situation. The $x$-axis (horizontal) can be arbitrarily chosen to be the axis of symmetry. Furthermore, we can arbitrarily choose the observation point to be only parallel the $x$-axis (for a given distance out along the $z$-axis) and co-planar with the $y=0$ plane. Each source on $x>0$ will be exactly the same distance away from our observation point as its counterpart on $x<0$. Therefore, we can simply double the contribution from each source, with the exception of those along the $x$-

Figure 2-5 Reconstructed FWM. $k=9$ m$^{-1}$, $z_p=6E-5$ m, 21×21 elements, element spacing=0.92mm
axis \((y=0)\), of which there is only one set of sources. The reconstructed FWM pulse can then be computed using:

\[
f_R(x, y, z, t) = -\sum_{m=-M}^{M} \sum_{n=-N}^{N} \frac{1}{4\pi R_{mn}} s_{FWM, m/n}(m\Delta x, 0, z', t - \frac{R_{mn}}{c}) \Delta x \Delta y
\]

\[
-2\sum_{n=1}^{N} \sum_{m=-M}^{M} \frac{1}{4\pi R_{mn}} s_{FWM, m/n}(m\Delta x, n\Delta y, z', t - \frac{R_{mn}}{c}) \Delta x \Delta y
\]

(2-11)

While Equation (2-10) requires \((2M+1)(2N+1)\) summations, Equation (2-11) only requires \((2M+1)(N+1)\) summations.

For comparison, consider a 21 x 21 array (elements go from \(x=[-10, 10] \times y=[-10, 10]\)). Using Equation (2-10) would require 441 summations, but Equation (2-11) uses only 231 summations. So using Equation (2-11) would take only 52% the computation time that Equation (2-10) would.
3. Experimental Setup

3.1. Project Goals

The purpose of this project is to construct a system for investigating acoustic localized waves. In preparing such a system for use, the pertinent parameters of this system must also be characterized. Although focused wave modes have potential applications in many areas, our main interest lies in medical imaging. Therefore, the experimental equipment will be designed to operate at frequencies most useful to medical ultrasonic imaging.

3.1.1. Specifications:

The following is a list of the main criteria which had to be considered for the specification of this system:

1. it should operate at frequencies of interest in medical imaging. (~1-10 MHz);
2. it should have a flexible configuration and
3. be constructed for a limited budget of ~$5000 for new equipment acquisitions.

3.2. System Design

Due to the requirement for a flexible system setup, a design based on synthetic aperture and synthetic field sampling is an obvious choice. A single degree of freedom motion stage was chosen. Although this does not limit the types of experiments that can be performed, it does require extra time to set up and operate the system.
3.3. Experimental Equipment:

This system design required the following pieces of major equipment:

(A Existing equipment)

1 IBM compatible i486 PC

1 PC D/A board [Gage CompuGen 840A]  

1 PC data acquisition board [Gage CompuScope Lite]  

1 PC step motor controller board [Keithley-Metrabyte MSTEP 3]  

1 single axis motion stage with 12” travel [DCI-12]  

1 stepper motor  

1 stepper motor driver and heat sink [API CMD-50]  

2 3.5 MHz, 0.25” diameter ultrasonic transducers [Panametrics V384]  

2 2 meter waterproof UHF to BNC cables  

1 high bandwidth receiver amplifier  

1 ~80 cm × 60 cm × 60 cm Plexiglas water tank
3.4. Experimental Setup

Figure 3-1 Overview of Experimental Setup

The CompuGen 840A, CompuScope Lite and MSTEP-3 are all installed in the i486 PC. Each card is attached to the appropriate device. The CompuGen D/A waveform generator drives one of the transducers at a signal level up to ±10 V. The CompuScope Lite data acquisition board receives signals from the other transducer, after amplification by a custom built 3-stage, programmable gain amplifier. Finally, the MSTEP-3 stepper motor controller is used to control the translation stage.
3.4.1.1 Gage CompuGen 840A

This D/A waveform generator board is equipped with 3 I/O connectors:

1) BNC trigger input;
2) pin digital output; and
3) BNC analog output.

The analog output generates signals between ±1 V, ±2 V, ±5 V and ±10 V corresponding to gain levels of ×1, ×2, ×5 and ×10. For most of the experiments, ×10 gain is required to obtain measurable signal levels. This D/A board is capable of 8 bit output at 40 Msamples/second. Therefore, it is suitable for generating signals up to a theoretical maximum of 20 MHz.

M.A.Sc. Thesis
Isaac Jing Herng Leung
06/07/96
3.4.1.2. Gage CompuScope Lite

The CompuScope Lite data acquisition board has 3 input BNC connectors:

1) trigger input;
2) channel A input; and
3) channel B input.

![Image of CSLITE I/O Ports, viewed from rear of PC.]

This A/D card is capable of sampling at a maximum rate of 40Msamples/s on a single channel or 20Msamples/s on both channels simultaneously. Full scale input voltage range on each channel is ±1 V or, when ×5 gain is specified ±200mV. For most of the experiments, one channel is set to ×1 gain and the other to ×5.


3.4.1.3. *Keithley-Metrabyte MSTEP 3*

I/O through this board is accessed through a 50 pin connector. Relevant connections are shown in the diagram and table.

The connector on the PC card is 3M (Scotchflex) #3433-5303. For ribbon (insulation displacement) cable, the mating connector is 3M #3425-6050.

![Diagram of MSTEP-3 Connector](image)

**Figure 3-4 MSTEP-3 Connector, viewed from rear of PC.**

Connections required between the MSTEP-3 controller card and the CMD-50 Stepper Motor driver are:

**Table 3-1 MSTEP-3 to CMD-50 connections.**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Source/Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A Axis Direction</td>
<td>(Output from card)</td>
</tr>
<tr>
<td>5</td>
<td>A Axis Step Pulse</td>
<td>(Output from card)</td>
</tr>
<tr>
<td>27</td>
<td>A Axis Limit 1</td>
<td>(Input to card)</td>
</tr>
<tr>
<td>28</td>
<td>A Axis Limit 2</td>
<td>(Input to card)</td>
</tr>
<tr>
<td>31</td>
<td>Ground</td>
<td></td>
</tr>
</tbody>
</table>
3.4.1.4. API CMD-50 Stepper Motor Driver

The CMD-50 Stepper Motor Driver has a total of 15 terminals. However, not all of the terminals are used for the current experiment. Inputs that are not connected are internally pulled up to logic HI (+5 Volts).

![CMD-50 Stepper Motor Driver and required connections.](image)

- **STEP IN** (left pin 3) connects to **MSTEP-3: A AXIS STEP PULSE** (pin 4)
- **CW/CCW** (left pin 4) connects to **MSTEP-3: A AXIS DIRECTION** (pin 5)
- **5VDC IN** (left pin 7) connects to a _+5 Volt power supply._
- **ROI** (left pin 8 and 9) connects to a _≈6.34kΩ resistor._
Chapter 3: Experimental Setup

- **COMMON GROUND** (left pin 10) connects to **MSTEP-3: GROUND** (pin 31) as well as most other grounds, including +5 V ground and +15 V ground but not Stepper Motor (black wire and white wire), which are tied together and left unconnected.
- **G/W** (right pin 5) connects to Stepper Motor: White with Green stripe wire.
- **GRN** (right pin 4) connects to Stepper Motor: Green wire.
- **RED** (right pin 3) connects to Stepper Motor: Red wire.
- **R/W** (right pin 2) connects to Stepper Motor: White with Red stripe wire.
- **+VOLTAGE SUPPLY** (right pin 1) connects to a +15 Volt power supply. The driver will support up to 40 Volts maximum. However, it is recommended that not more than 20 Volts input is used.

In addition to the electrical connections, the heat sink for the CMD-50 is required in order to keep the motor driver within safe temperatures during operation.

![CMD-50 Stepper Motor Driver](image)

**Figure 3-6 CMD-50 Stepper Motor Driver with installed heat sink.**
Figure 3-7 shows a schematic of the connections required for controlling the translation stage with the PC.

![Figure 3-7 Connections to control translation stage.](image)

**3.4.1.5 DCI 12” Motion Stage and Water Tank**

The motion stage is driven by a single 2.25 Volt, 4.6 Amp stepper motor. The motor is capable of both full step and half step operation. The lead screw on the motion stage has a claimed repeatable accuracy of 0.0008 in. (0.00002032 m) and is specified to move the motion stage at 0.00508 m / revolution.

The water tank measures approximately 80cm × 60cm (viewed from the top). However, the motion stage is not long enough to span the 60cm width of the water tank.
Chapter 3: Experimental Setup

Therefore, the motion stage is first mounted on aluminum spars long enough to span across the length of the water tank.

The mount for the receiving transducer is attached to the metal spars of the motion stage. By mounting the receiving transducer directly on the base plate, motion from the surrounding environment is effectively negated as the source and receiver will maintain their relative positions.

The motion stage is positioned across the length of the water tank at a position ~20cm from one edge. This is done to maximize the time required for any (unwanted) reflected signals to arrive at the receiving transducers.

Let us define an experimental region within the tank, approximately $30 \times 20 \times 20$ cm in size and centered within the tank. The length of the transmitted pulses are on the order of 50 $\mu$s or less. If the transmitter and receiver are position at opposite ends of the experimental region, we can then calculate that the total time of flight (TOF) required to transmit the pulse and the time required to receive the entire pulse is approximately 200 $\mu$s.

Considering the same situation, if we calculate the TOF for pulses reflected off the tank walls, we can find that it will require approximately 400 $\mu$s for these reflected pulses to arrive at the receiver. Therefore, it is quite certain that unwanted reflections will not be received along with the direct pulse and sound absorbent material on the water tank walls is not required.
Figure 3-8 Motion Stage setup and Time-of-Flight estimates.
3.4.1.6. **Panametrics 3.5MHz Ultrasonic Transducers**

2 ultrasonic transducers are used in this system. One is as the pulse generator and one as the receiver. The operational characteristics are nearly identical, as expected.

The transducers are immersion transducers originally designed for use in Non-Destructive Testing (NDT) applications. Operating frequency is centered at ~3.5MHz. The active diameter is 0.25" and they are designed to unfocused. The Panametrics transducers are designed to be operated using a large negative going pulse (in order avoid overheating the transducer and to avoid depoling the piezoelectric crystal) with the maximum energy not exceeding 125mW. Although negative going pulses are not used in these experiments, the pulse power levels are low enough such that damage is not expected to occur to the transducers. The transducers are typically designed to use pulses of several hundreds of volts and milliseconds in duration.

Using the technical notes supplied by Panametrics, we can obtain an estimate of the maximum voltage that can be applied to the transducers or our particular operating conditions. Three equations are used in this estimation:

![Panametrics Ultrasonic Transducer 3.5 MHz 0.25" diameter (unfocused)](image)
Chapter 3: Experimental Setup

\[ V_{\text{rms}} = \frac{1}{2} (0.707)V_{\text{peak-peak}} \]  \hspace{1cm} (3-1)

\[ P_{\text{total}} = \frac{(\text{DutyCycle})(V_{\text{rms}})^2 \cos(\text{PhaseAngle})}{|Z|} \]  \hspace{1cm} (3-2)

\[ N_{\text{burst}} = \frac{(\text{Frequency})(\text{DutyCycle})}{\text{Rate}} \]  \hspace{1cm} (3-3)

where: 

- \( V_{\text{peak-peak}} \) is the peak to peak voltage;
- \( V_{\text{rms}} \) is the RMS voltage;
- \( P_{\text{total}} \) is the total power;
- \( Z \) is the impedance at the transducer input;
- \( N_{\text{burst}} \) is the number of cycles in a burst; and
- \( \text{Rate} \) is the rate of repetition (of bursts).

For this particular experiment, we know the following values: \( \text{Rate} \leq 1 \text{ Hz} \); \( N_{\text{burst}} = 1 \), \( |Z| = 50 \ \Omega \) and \( 0.5 \text{ MHz} \leq \text{Frequency} \leq 10 \text{ MHz} \).

Using Equation (3-3), and using values to maximize the DutyCycle, we can calculate that \( \text{DutyCycle} = 2 \times 10^{-6} \text{ s/s} \). Now, using Equation (3-2), assuming maximum allowed power (125 mW) and \( \cos(\text{PhaseAngle}) = 1 \), we can calculate that \( V_{\text{rms}} \approx 1767 \text{ V} \). Finally, using Equation (3-1), the maximum allowable value for \( V_{\text{peak-peak}} \) is approximately 5000 V. The maximum capability of our system is 20 Volts peak to peak, so we are well under the design limits.
3.4.1.6.1. Initial preparation of the transducers for use

The transducers are connected to the data generation and acquisition equipment via waterproof UHF-BNC cables. These are regular UHF-BNC cables which have the UHF end rendered waterproof using a silicone based sealant.

The system was quickly assembled for a few initial tests to make sure that all components were working and to fix major problems, if any, with the equipment setup. Figure 3-10 shows a diagram of the experimental setup. A 50Ω terminator was connected output line to ensure a proper output load matching for the D/A generator.

![Diagram of experimental setup](image)

Figure 3-10 Schematic of Experiment setup.

3.4.1.6.2. Signal Generation and Reception Accuracy

The first test was to determine if the CompuGen 840A would be capable of accurately reproducing the desired signals. The CompuScope Lite was used to record the signal directly from the CompuGen 840A’s output. Both boards were operated at 20 Msamples/second.
Figure 3-11(a) shows a \( \text{sinc}(t) \) signal with a theoretical 2MHz bandwidth. 256 points were used in the calculated signal. Figure 3-11(b) is the received signal with no averaging and Figure 3-11(c) shows a 20 mean average of the received signals.

![Graphs of calculated and received signals](image)

Figure 3-11 2MHz bandwidth \( \text{sinc}(t) \) signal.
Chapter 3: Experimental Setup

Figure 3-12 shows another \( \text{sinc}(t) \) signal. Again, 256 points were used to calculate the signal. This time, the theoretical bandwidth of the signal is 8MHz. This is close to the Nyquist frequency for 20MHz sampling, so the signal reproduction is not expected to be as good as that of a 2MHz signal.

![Diagram of calculated and received signals](image)

Figure 3-12 8MHz bandwidth \( \text{sinc}(t) \) signal.
Chapter 3: Experimental Setup

In addition, two FWM source pulses were arbitrarily chose for the test (the relevant parameters are \( k = 12 \, m^{-1}, z_0 = 6 \times 10^5 \, m \)). Figure 3-13 shows the FWM source at \( r = 0 \, mm \) and Figure 3-14 shows the FWM source pulse for \( r = 9 \, mm \).

3.4.1.6.3. Driving the ultrasonic transducer

The next test was to determine the ability of the CompuGen 840A to drive the ultrasonic transducers. The output signal is again measured with CompuScope Lite directly connected to the D/A output. Both boards are operated at 20M samples/second. The output signal was recorded as in the previous test, and also with the transducer connected directly to the D/A output. The D/A output was set to maximum output levels of ±1 Volt.
Chapter 3: Experimental Setup

The first test signals are $sinc(t)$ signals, which were used in the previous test. Figure 3-15 shows a 2MHz bandwidth signal. At this frequency, the CompuGen 840A appears to have no trouble driving the transducer at this voltage level.

![Graphs of calculated and received signals with and without transducer](image_url)

Figure 3-15 2MHz bandwidth signal with transducer.
Figure 3-16 shows the result from a 8MHz bandwidth $sinc(t)$ signal. At this frequency, the signal reproduction is clearly inferior to that of a 2MHz signal.
FWM source functions were also tested with and without the load of the transducer.

Arbitrary source functions were again chosen. However, this time, the relevant parameters
Chapter 3: Experimental Setup

used \((k=9 \text{ m}^{-1}, z_0=6 \times 10^{-4} \text{ m})\) were selected with the aid of simulations and taking into account the performance limits of the translation stage. The values selected are intended to be used in an actual, experimental measurement.

Figure 3-17 shows the FWM source at \(r=0 \text{ cm}\) and Figure 3-18 shows the FWM source at \(r=1.8 \text{ cm}\). Both are reproduced fairly accurately, even when loaded with the transducer.

The same test was repeated for gain levels of \(\times 2, \times 5\) and \(\times 10\). At these higher output levels, a voltage divider (using a 10k\(\Omega\) POT) was used to scale down the signal levels to \(\pm 1\) Volt so that it could be recorded using the CompuScope Lite. The results were similar to the
×1 case, except at ×10. At a gain setting of ×10, the output waveform was slightly distorted, regardless of whether the transducer was connected or not.

Figure 3-19 and Figure 3-20 show the FWM sources at a ×10 gain. Although the waveform is distorted, the CompuGen 840A is able to reproduce approximately the same waveform, with or without the transducer connected.

Figure 3-19 FWM source at r = 0 cm, ×10 gain.  

Figure 3-20 FWM source at r = 1.8 cm, ×10 gain.
4. Experimental Results

All following experiments were performed using the equipment set up shown in Figure 3-10 with the addition of the connection between the PC and the translation stage.

4.1. Amplitude Decay with Distance

4.1.1. Background

All real sources have a characteristic near field and far field distance. In the far field, the source appears and behaves like an infinitely small point source radiator. In the near field, this is not the case. Spherical wave fronts will not be observed and the wave amplitude does not decrease inversely proportional to the distance from the source, as would be seen in the far field. In fact, the amplitude of the signals will remain relatively constant in the near field.

The length of the near field of any particular source can be closely approximated by the following equation (Panametrics Technical Notes):

\[ d_N = \frac{D^2}{4\lambda} \left[ 1 - \left( \frac{\lambda}{D} \right)^2 \right] \]  

(4-1)

where:  
\( d_N \) = length of near field  
\( D \) = diameter of source  
\( \lambda \) = wavelength of the source signal

This equation is empirically derived for ultrasonic transducers. It is valid only when \( D \gg \lambda \). The exact equation is quite complex and Equation (4-1) suffices for an estimate.
Figure 4-1 shows a plot of the near field variation with frequency for the Panametrics transducers which have an active diameter of approximately 6 mm. This plot is obtained using Equation (4-1) and the relation:

\[ c = \lambda v \]  

(4-2)

where:

- \( c \) = speed of sound in the medium (~1540 m/s in water at 20°C);
- \( \lambda \) = wavelength; and
- \( v \) = frequency.

In practice, this increase in near field length does not occur like this. This is because the acoustic impedance of a particular medium varies with frequency. In the case of water (or animal tissue), the acoustic impedance increases rapidly as frequency is increased and signal...
amplitude begins to drop off earlier for high frequencies than would be predicted by Equation (4-1). In addition, any real signal contains a spectrum of frequencies and the actual near field length is a combination of the results from all the frequencies contained in the signal.

4.1.2. Measurements

Although equation (4-1) is accurate, in practice it can be difficult to apply to any detailed measurements. This is partly due to the usual uncertainties in measured values. Most of the problem, though, is in producing a source signal with a single, unique frequency. This is generally not possible and any real source has some spread, however small, of frequencies. In theory, this can be accounted for and the signal amplitude behaviour can be more accurately described. However, this can be computationally intensive. As a result, this equation is simply used as an estimator to measure the accuracy of measured results.

This series of experiments uses the ultrasonic transducers as a transmitter/receiver pair. Measurements were taken at 1mm increments and the value at each point was taken to be the mean over 10 readings at that point. The purpose being simply to reduce random noise.

The frequency of a particular signal, for computational purposes, was taken to be the average frequency of the signal. The average frequency was calculated using the following equation (used for discrete measurements):

\[
f_{avg} = \frac{\sum_{n=1}^{N} f_n |X_n|}{\sum_{n=1}^{N} |X_n|}
\]

where: \( f_{avg} \) = average frequency of the signal
Chapter 4: Experimental Results

\[ f_n = \text{frequency}, f_n = n/T, n=1,...,N, T = \text{time length of the signal}. \]
\[ X_n = \text{the Fourier coefficient at frequency } f_n \]
\[ N = \text{number of points for which } f > 0 \]

4.1.2.1. Decay at 3.5MHz

The first set of measurements used a single sine cycle as input with an \( f_{\text{avg}} \approx 3.7 \text{ MHz} \). The characteristics of this signal are plotted in Figure 4-2.

The actual received signal has different characteristics. This signal has \( f_{\text{avg}} \approx 3.5 \text{MHz} \) and its characteristics are show in Figure 4-3.

Using Equation (4-1) and \( f=3.5\rightarrow3.7 \text{ MHz} \), we can estimate that the near field distance should be \( \sim 2.0\rightarrow2.2 \text{ cm per transducer} \). Therefore, for the transmitter-receiver pair, the near field distance should be roughly twice that, or approximately \( 4\rightarrow4.4 \text{ cm} \).
Signals were recorded from ~0 to 20 cm from the source in 1 mm increments. At each measurement point, only the maximum point of the detected signal was recorded. The final data consists of 200 points representing the maximum signal amplitude at each point.
Chapter 4: Experimental Results

The results are plotted in Figure 4-4. The dashed lines are the asymptotes for constant amplitude and $1/r$ (where $r$ represents the transmitter-receiver distance) amplitude decay. Using the -3dB point, the near field can be estimated to be $\sim 4.5$ cm, which is in approximate agreement with the theoretical result.

![Signal Amplitude Decay (-3.5 Mhz)](image)

Figure 4-4 Normalized Amplitude Decay vs. Distance

4.1.2.2. Decay at 2.5 MHz

The 2nd set of measurements used a signal with average frequency of approximately 2.5 MHz. The input signal is again a single cycle of a sine with a peak frequency of
approximately 2 MHz, but $f_{avg} = 2.5$ MHz. The characteristics of this signal are plotted in Figure 4-5.

The received signal characteristics is shown in Figure 4-6 and has the transducer frequency response imposed on it so that the $f_{avg} = 2.9$ MHz, which is also close to the peak frequency.

Using these values, we can calculate the near field distance for a single transducer to be $\approx 1.5 \rightarrow 1.7$ cm. For the entire system then, the near field length should be $\approx 3.0 \rightarrow 3.4$ cm.

Measurements were taken as in the previous experiment, recording the maximum signal amplitude in 1 mm increments out to 20 cm from the source. The results of the
experiment are plotted in Figure 4-7. Again, using the -3dB point as an estimator, we can see that the near field distance is \( \approx 3.5\text{cm} \).

\[ \text{Amplitude Decay vs. Distance (~2.5 MHz)} \]

![Amplitude Decay vs. Distance (~2.5 MHz)](image)

Figure 4-7 Amplitude Decay with Distance for \( \approx 2.5 \text{ MHz} \) signal.

### 4.1.2.3. Decay of FWM Source Pulse

For the third measurement, a different pulse was used. This pulse is one of the source pulses required to generate a FWM pulse. This particular signal is the signal required for the central source of an FWM generating array. The parameter values are \( k=12 \text{ m}^{-1}, z_0=6\times10^{-5} \text{ m} \).

The characteristics for the generated and received pulse are shown in Figure 4-8 and Figure 4-9, respectively. For the generated pulse, average frequency is calculated to be \( f_{\text{avg}} \approx 1.8 \text{ MHz} \) while the received signal has \( f_{\text{avg}} \approx 2.6 \text{ MHz} \).
Using these values, we can then calculate the expected near field length, for the entire system, to be approximately 2–3 cm, probably close to 3 cm. The experimental data is plotted in Figure 4-10. The observed near field length is approximately 4 cm, which is in close agreement with the theoretical value.

Figure 4-8 Transmitted signal characteristics

Figure 4-9 Received signal characteristics
4.2. Amplitude Decay with Angle

4.2.1. Background

The amplitude response with angle of the transmitter/receiver pair was also measured. For an ideal point source, the signal amplitude does not vary with the angle between the source and the receiver. It should only vary with the distance between the source and the receiver. However, these are real transducers with an element diameter of ~6mm and the signal amplitude will exhibit some sort of angular dependence.
In practice, the width of the beam cannot be smaller than the diameter of the transducer. For a circular radiating transducer, a commonly used first approximation to model the spread of the beam is to assume that the beam is well confined within a cylinder of radius $D/2$ up to the distance $z_F$. Thereafter, it is assumed that the beam diverges within a cone of apex angle $\theta_T$. This angle is calculated using:

$$\sin\left(\frac{\theta_T}{2}\right) = \frac{0.619}{\lambda} \frac{D}{2}$$

(4-4)

where: $\theta_T$ is the spread angle at which the beam is -6db from its central value;

$\lambda$ is the wavelength; and

$D$ is the transducer diameter.

For these particular transducers, we can calculate that $\frac{\theta_T}{2} = 5.2^\circ$ at 3.5 MHz. Using this result, we can also determine that $z_F = 3.3 \text{ cm}$. Therefore, for distances less than $z_F$, we can determine the spread angle as $\theta = \tan^{-1}\left(\frac{D}{2z}\right)$. For distances greater than $z_F$, the spread angle will be $\theta = \frac{\theta_T}{2}$. 
Chapter 4: Experimental Results

4.2.2. Measurements

Measurements were taken at various distances of transmitter-receiver separation. The purpose of this experiment is to verify that they transducers behave approximately as predicted by theory, and to document the operational characteristics for future use.

For each separation distance, the appropriate perpendicular and parallel distances were calculated to achieve the desired transmitter-receiver angle. This is easily calculated with:

\[ d_\parallel = R \cos(\theta) \]  \hspace{1cm} (4-5)

and

\[ d_\perp = R \sin(\theta) \]  \hspace{1cm} (4-6)

\( d_\parallel \) was achieved by using the motion stage to move the appropriate distance. The motion stage has a claimed repeatable accuracy of ±0.0008 in. \( d_\perp \) was moved manually using a scale. Repeatable accuracy is approximately ±0.5 mm.
4.2.3. $R = 3$ cm

The first set of measurements was made at a transmitter-receiver distance of 3 cm. A signal with average frequency of ~ 3.5MHz was used, the same signal as was used in measuring the amplitude decay with distance (single cycle of sine wave). At this frequency and distance, the signal is being measured within its near field distance.

The data plotted in Figure 4-13 results from the mean of 20 measurements at each point. Measurements were taken in $2^\circ$ increments until $20^\circ$, at which point measurements occurred in $10^\circ$ increments.
4.2.4. \( R = 12 \) cm

A similar measurement was made at a source-receiver distance of 12 cm. At this distance and frequency, we should be in the transition between the near and far field. Figure 4-14 shows the measured results. Although the amplitude still varies with angle, one can see that the amplitude response with angle is becoming more spread out, relative to that at \( R = 3 \) cm.
4.2.5. \( R = 20 \text{ cm} \)

Further measurements were taken at a source-receiver distance of 20 cm to see if the response differs greatly from that at \( R = 12 \text{ cm} \). At this distance, the signals measured should be well into the far field. These results are plotted in Figure 4-15. As you can see, it does not vary very much from the response at \( R = 12 \text{ cm} \). Past angles of \( \pm 10^\circ \), the signal amplitude falls off rapidly.
Figure 4-15 Amplitude response vs. Source-receiver angle at R=20cm.

Although the transducers do not exhibit point source behaviour at distances up to 20 cm apart as we would like, this does not prevent us from performing future experiments. For this synthetic array system with only 1 axis of motion, the amplitude response with angle need not be accounted for as the transmitter-receiver pair are always directly facing one another (0°).

**4.3. Signal Reconstruction**

A method of reconstructing the transmitted signal is necessary for future experiments. The received signal will be corrupted by noise and the non-ideal performance of the
experimental equipment. Processing of the received signal is required in order to recover the original, transmitted signal.

4.3.1. Reconstruction by integration

4.3.1.1. Background

Previous work has shown that the equivalent system response for transmitting and receiving an acoustic wave from a piezoelectric can be described by (Duck, 1977):

$$H_s(\omega) = H_r^2(\omega)$$

where:

\(\omega\) denotes frequency;

\(H_s(\omega)\) is the transfer function describing transmitting and receiving;

\(H_r(\omega) = \frac{v(\omega)}{F(\omega)}\);

\(F(\omega)\) is the driving function of the piezoelectric crystal; and

\(v(\omega)\) is a disturbance in the propagation medium (the propagating signal).

Furthermore, from the Mason or KLM model of a piezoelectric acoustic transducer, we know that:

$$V(\omega) \propto \frac{1}{j\omega} v_p(\omega)$$

where:

\(V(\omega)\) is the voltage on the piezoelectric; and

\(v_p(\omega)\) is the velocity of the crystal surface.
Chapter 4: Experimental Results

In the ideal case, the crystal velocity, \( v_p(\omega) \), is directly related to the fluid particle velocity and hence the wave propagation, \( v(\omega) \), in the water. Using this, we can determine that \( H_p(\omega) = j\omega \) and therefore \( H(\omega) = (j\omega)^2 \).

Given a known driving function \( V_{in}(\omega) \), we can then (neglecting some constant multiplicative factors) describe the received the received signal \( V_{out}(\omega) \) as:

\[
V_{out}(\omega) = (j\omega)^2 V_{in}(\omega)
\]  

(4-9)

In the time domain, this then corresponds to two differentiations. So, we can theoretically recover \( v_{in}(t) \) knowing only \( v_{out}(t) \) by using:

\[
v_{in}(t) = \int \int v_{out}(t) dt
\]  

(4-10)

Observing the plot of the system frequency response, one can see that the experimental result seems to be a good match to the theory. Therefore, we should be able to calculate and reconstruct the original signal by simply integrating (twice) the received signal.

4.3.1.2 Measurements

The first signal is a FWM source function at location [8,0] in the source array. The relevant parameters for this pulse are \( k=5 \text{ m}^{-1} \), \( z_0=4\times10^{-4} \text{ m} \) and source element spacing of 1.8 mm. The received signal is averaged over 30 separate readings.

Figure 4-16 shows the calculated and received signals. As well, the calculated result has been differentiated twice to compare with the received value.
Chapter 4: Experimental Results

Since the theoretical results appears to be a good match to the received signal, we proceeded to integrate the received signal. Figure 4-17 shows the results from one and two consecutive integrations.

Figure 4-16 FWM source at [8,0].
As can be seen in the Figure 4-17(top), the first integration exhibits a slope in the result, which manifests itself as a curve in the second integration. At first glance, this appears to be the result from a small DC offset in the original signal. Correction of this error is then simply achieved by including the value of this offset in the integration.
However, a few simple trials can show that this is not the case. Figure 4-18 shows the result of including a constant offset in the integration. In this case, the offset value has been calculated by obtaining the mean value of the received signal.

Figure 4-18 Integration of received signal + offset.
4.3.1.3. Curve Fitting the Offset

To solve this problem, we accounted for the varying offset by fitting a polynomial curve to this and subtracting this off before integration. The polynomial curve fit was calculated using MatLab's `polyfit`, which calculates the polynomial coefficients and `polyval` which calculates the result given the coefficients and the dependent variable (for more details, refer to Appendix B: `processa.m`). The final result is shown in Figure 4-19.

![Received signal after offset correction and integration](image)

Figure 4-19 Received signal after offset correction and integration.

M.A.Sc. Thesis

Isaac Jing Herng Leung

06/07/96
This method of reconstruction was then applied to several other signals to test its effectiveness. The results are shown in the following graphs:

**Figure 4-20 FWM source at [0,0].**

**Figure 4-21 FWM source at [10,0].**

### 4.3.1.4 High Pass Filtering to Remove Offset

As the unwanted offset is an apparently low frequency signal in comparison to the signal waveform, another possible method of correcting this problem is through high pass filtering of the signal.

A FIR filter using a Bartlett window was implemented using MatLab’s built in functions `fir1` and `bartlett`. After some experimentation the lower cutoff frequency was
set at 300 kHz. As an initial test, the FWM source pulse at location [8,0] in the array was used. The parameters are the same as in the previous section. Figure 4-22 shows the result.

This method of processing was then used on several other FWM source pulses so that the results could be compared with the previous technique. The results of this are shown in the following two figures.
4.4. Reconstruction from Synthetic Array

Reconstruction of a pulse from a synthetic array source is relatively simple, in principle. In order to reconstruct the pulse, one needs to record the signals from each of the separate sources. Then, using the methods developed above, each individual pulse is processed to determine the 'actual' signal. Finally, the results of this are all summed together to produce the final result.

For this experiment, the system was used to generate, and record the necessary signals for launching a FWM pulse. Measurements were taken at a receiver-central source distance
Chapter 4: Experimental Results

of 12 cm. The size of the array is $21 \times 21$ elements and the spacing between elements is taken to be 1.8 mm. The relevant parameters for the FWM pulse are $k=5 \text{ m}^{-1}$, $z_0=4 \times 10^{-4} \text{ m}$.

Figure 4-25 shows an ideal FWM pulse at a distance 12 cm from the origin. It has been simulated with the same number of points and resolution as the experimental pulse. The experiment records 256 points along the axis of propagation and 16 sets of measurements are recorded at equally spaced distances from the central propagation axis up to a maximum of 3 cm out. Figure 4-26 shows a simulated FWM pulse, reconstructed at 12 cm from the source. The simulated source uses the same parameters as the experimental pulse. It uses $21 \times 21$
sources separated by 1.8 mm and $k = 5 \, \text{m}^{-1}$, $z_0 = 4 \times 10^{-4} \, \text{m}$. Again, the reconstructed resolution has been set to the same as that of the experimental pulse.

For the experimental pulse, 256 points of data were collected from each source. Further, 16 equally spaced sets of measurements were taken out to 3 cm radially out from the axis of propagation. The data was processed in Matlab (see Appendix B, `m_create.c`) to give the result shown in Figure 4-27 and Figure 4-28. Figure 4-27 shows the result of processing using the polynomial curve fit method described previously. Figure 4-28 shows the result using the high pass filtering method.
Chapter 4: Experimental Results

Figure 4-27 Experimentally reconstructed pulse. Curve fit correction.

Figure 4-28 Experimentally reconstructed pulse. High pass filtering.
As is expected, the two methods give very nearly the same result. The experimental results provide an excellent match to the theoretically reconstructed result. A ‘top’ view, with amplitude coded by grey scale, of the same plots are shown below. Figure 4-29 is the theoretically reconstructed pulse. Figure 4-30 and Figure 4-31 show the experimental result resulting from each of the two different signal reconstruction methods. These plots further highlight the similarity between the experimental and theoretical results.

Figure 4-29 Theoretically reconstructed pulse. Greyscale coded amplitude plot. (Top view of pulse)
Figure 4-30 Experimental result (curve fit correction) Greyscale coded amplitude plot. [Top View].

Figure 4-31 Experimental result (High pass filtering), Greyscale-coded amplitude plot. [Top View].
5. Conclusions and Recommendations

5.1. Results

The goal of this project was to design, construct and characterize a system for launching localized waves. In order to save costs and provide flexibility in configuration, the system is designed for synthetic array generation of localized wave sources.

The main components of this system consist of:

- Immersible, ultrasonic transducer pair for transmitting and receiving.
- High frequency signal amplifier.
- Computer controlled motion stage.
- Computer controlled signal generator.
- Computer controlled data acquisition hardware.

All software has been written to control and acquire data for various experiments. Most of the relevant software is written in C and can be easily upgraded for new experiments. In addition, Matlab signal processing routines have been created for use in these experiments.

Several experiments were conducted in order to characterize the system for future use. Measurements were obtained to evaluate the signal amplitude decay with distance, and to determine the angular response of the transducer pair. In addition, an algorithm was devised to process received signals to recover the original transmitted signals. Finally, the system was tested in the generation and reconstruction of a sample FWM pulse.

The signal amplitude decay with distance agrees very closely to the predicted result. In theory, the signal amplitude will remain constant until the end of the near field distance of...
Chapter 5: Conclusions and Recommendations

the source. At which point the signal amplitude should decay inversely with the distance.

Several sets of measurements were taken, with source signals having various average and peak frequencies. Using calculated information on signal frequency and known data on the size of the transducer, the near field length was calculated. The experimental results agree closely with the theoretical results. Beyond the near field length, the amplitude decays as $1/r$, while within the near field length, the amplitude stays roughly constant. Near the near field-far field boundary, the experimental results exhibits a smooth roll-off to join the two regions.

Using a simple theoretical model, the beam spread is predicted to be confined within a cone beyond approximately 4 cm. The experimental results showed characteristics which are similar that which is predicted. The angle at which signal amplitude falls to half the central value appears to be approximately constant (~7°) at transmitter receiver distances from 6 cm up to 20 cm.

In order to properly conduct future experiments, it is necessary to account for the system’s effect on the signals. We need to be able to reconstruct transmitted signals from the received signal.

Theory predicts that the received signal in such a system will be the second time derivative of the transmitted signal. This was found to be true. However, there was one major problem in processing the received signals due to some slowly time varying signal imposed on our received signal. Two methods were developed to correct for this.

This first procedure involves correcting for a lower frequency signal by using a polynomial curve fit. This curve is then subtracted from the received signal before further processing. This corrected signal is then integrated twice to recover the original signal.
The second method involves using a high pass filter to remove the unwanted low frequency signal. A Bartlett windowed FIR filter is used to remove the low frequency signals. The result is then integrated twice to recover the original signals.

As a final test of the system performance, a test experiment was run in an attempt to generate and reconstruct a sample FWM pulse. This was done at a simulated distance of 12 cm from the FWM source array. The source array used $21 \times 21$ sources each set at equal spacings of 1.8mm apart. The experimental result was compared with the expected, theoretical result.

Although this does not confirm the launch or propagation of a FWM pulse, it does show the capability of this system to launch a wave from a synthetic array source and reconstruct a result. Hopefully, future experiments will be able to confirm the launch of a FWM pulse and to measure its characteristics.

5.2. **Recommendations**

At the present time, the experimental system is in complete operation. All hardware and software is available to run at least some rudimentary localized wave experiments. Possible experiments include:

- Generation and reconstruction of FWM pulses.
- Measuring Amplitude decay with distance of such pulses.
- Measuring the degree of localization of a FWM pulse as a function of distance.

Software is completely functioning. However, improvements in ease of use and speed of execution would be beneficial. Currently, some options require complete re-compilation of
Chapter 5: Conclusions and Recommendations

the code in order to change them. As well, an entire experimental run for generating a single FWM pulse requires approximately 9 hours for a $21 \times 21$ source array and 16 sets of measurements taken radially. This has already been decreased from the original requirement of 20 hours through careful programming and optimization.

The hardware for this system is also in functioning order. Again, improvements can also be made in ease of use.

The signal amplifier is sufficiently useful at the current time. However, an ideal amplifier would have a larger bandwidth and also have continuously variable, computer controllable gain settings. Although this may not be achievable, a new amplifier incorporating some of these enhancements would increase the usability of the current system.

Much of the experimental equipment can also be further improved by increasing the compactness of the setup. Steps can be taken to use smaller equipment casings, shorter wire leads and more compact connectors. Not only does this increase visual appeal, but it will also lessen the chances of accidents. By careful placement of equipment and shortening of wire leads, chances of accidentally tripping or catching on some loose wires are greatly reduced.

Although noise levels are not currently a large problem with this system, proper equipment shielding and other noise reduction techniques would go further towards increasing the accuracy and reliability of this system. One rare bug that occurs in this system is that a trigger signal is missed during an experiment and the system is halted, waiting for the trigger. Noise is a suspect in this problem.

Significant improvements can be made by acquiring faster and better signal generators and acquisition equipment. At the current time, the equipment can generate and measure
Chapter 5: Conclusions and Recommendations

signals of up to 10MHz. However, this is only the theoretical limit imposed by the Nyquist frequency. In practice, accurate measurement and generation of signals are limited to those signals with frequencies much lower than 10 MHz. The system current is designed to operate near 1-2MHz for the generation of FWM source pulses. However, this is not close enough to the central frequency of the ultrasonic transducers which have a peak response at 3.5MHz. As a result, efficiency of the system is reduced.

Although the current system does have some flaws and many improvements can be made, it has been shown that the equipment functions properly and is capable of carrying out at least simple localized wave experiments.
Bibliography


Appendices

- Appendix A: Signal Amplifier
- Appendix B: Program Listings
Appendices

Appendix A: Experimental Equipment

A.1 Signal Amplifier

*Design and construction by Dr. M.R. Palmer*

**A.1.1 Amplifier Design**

The non-inverting amplifier consists of 3 major stages with selectable gain levels. The schematics for the amplifier are shown below.

![Amplifier Stage](image)

*Figure A-1 Amplifier Stage 1*
Figure A-2 Amplifier Stage 2

Figure A-3 Amplifier Stage 3
A.1.2 Amplifier Performance Characteristics

In order to characterize the amplifier performance, a broadband signal was used as the test signal. In this case, an inverted \( \text{sinc}(t) \) function was used. This particular \( \text{sinc}(t) \) function has a bandwidth of \( \sim 9.5 \) MHz and was truncated using a rectangular window.

The amplifier was set to \( \times 10 \) gain level (the lowest available) for all the measurements. In order to avoid saturation of the output, the input signal from the Gage CompuGen 840A was passed through a 10k\( \Omega \) potentiometer voltage divider.
Figure A-5 Voltage Divider for limiting the input signal amplitude during testing

The following graphs shows the results of the characterization:

Figure A-6 Input and Output Signals
Figure A- 6 is a plot of both the input signal and the output signal. Figure A- 7 shows a plot of the magnitude and phase response of the amplifier at the $\times 10$ gain setting.

Notes:

[1] The displayed amplitude response over 9.5 MHz should be ignored as the input signal's bandwidth is only 9.5 MHz.
[2] The phase response merely indicates the constant time delay performance of the amplifier. It is not an indicator of the absolute time delay. The signal must propagate through the entire data collection system and the exact value of the time delay is very dependent on when the trigger occurs and what section of the collected data is considered. For example, considering as little as 5 data points earlier or later can cause the absolute value of the time delay to vary by as much as an order of magnitude.

[3] Certain signals may be distorted by the POT voltage divider, particularly if the signal is high amplitude and frequency. This should be visually obvious on an oscilloscope as the attenuation is gradually increased. The best way to attenuate the signal is to get a proper attenuator for a co-axial line. If that is not available, a 10× oscilloscope probe will also serve as a quick and temporary solution.
Appendices

Appendix B: Program Files

B.1 C Program Files

This section contains the C files used in this project. The C programs controlling the Gage CompuGen 840A and CompuScope Lite utilize C drivers supplied by Gage. These are, respectively, `CG840DRV.C` and `CSLITDRV.C`. In addition, `TIMING.OBJ`, a pre-compiled file also supplied by Gage is required.

`840_ctrl.h`

/*****************************/
/* Header file for 840_CTRL.C */
/* Control routines for */
/* CompuGen 840A */
/*****************************/
#define SOFTWARE 0
#define EXTERNAL 1

int initialize_cg840 (int MODE, int SINGLE_SHOT, int GAIN, int RATE, int TRIGGER, int FILTER);
int calculate_digital_trigger (cg840_buffer_type output_buffer);
int calculate_sinc (int points, float time_start, float time_end, float freq, int FILE_OUT, cg840_buffer_type output_buffer);
int calculate_sin_t2 (int points, float time_start, float time_end, float freq, int FILE_OUT, cg840_buffer_type output_buffer);
int calculate_sine (int points, float time_start, float time_end, float freq, int FILE_OUT, cg840_buffer_type output_buffer);
double sqr(double x);
int convert_float_to_digital(double dpoint, double signal_max);
float convert_digital_to_float(byte digital_val, double signal_max);
double calculate_fwm_point (float t, float spacing, int X, int Y, float k, float z0);
int calculate_fwm_pulse (int points, float time_range, float spacing, int X, int Y, float k, float z0, cg840_buffer_type output_buffer, double *Norm_Factor, int FILE_OUT);
int output_waveform (int DIGITAL_TRIG, cg840_buffer_type output_buffer);
/* END */
/* ------------------------------ CompuGen 840A_Control ------------------------------ */

#include <conio.h>
#include <math.h>
#include <process.h>
#include <stdio.h>
#include "cg840drv.h"
#include "840_CTRL.H"
#include "DELAYs.H"

#define VERSION 0.98
#define pi 3.14159265358
#define c 1540
#define SOFTWARE 0
#define EXTERNAL 1
#define samples_per_cycle 128
#define bits 8
#define CG840_DEFAULT_CONFIG_FILE "CG840.INC"
#define DELAY 10

int initialize_cg840 (int MODE, int SINGLE_SHOT, int GAIN, int RATE,
int TRIGGER, int FILTER)
{
    clrscr();
    printf("************************************************************
" );
    printf("* Initialize CompuGen 840A V%3.2f *
", VERSION);
    printf("* * * * * -k -k ****************************") .
    printf("result_840 = cg840_driver_initialize (CG840_DEFAULT_CONFIG_FILE);
    if (result_840 == 0)
    {
        printf (" No CompuGen 840 boards were found.\n"");
        printf(" An error occurred!\n");
        printf(" Press a key to end...\n");
        getch();
        exit (1);
    }
    else
    {
        if (result_840 < 0)
        {
            printf (" Not all CompuGen 840 boards were found.\n");
            printf (" Number of CompuGen 840 boards found = %d\n", -result_840);
printf ("An error occurred!\n");
printf ("Press a key to end...\n");
getch();
exit (1);
}
/* Okay, everything seems to be alright with the 840A */
else
{
printf ("Number of CompuGen 840 boards found = %d\n",
result_840);
printf ("\n");
result = TRUE;
}
}
for (i = 1 ; i <= result_840 ; i++)
{
printf ("Using board number: %d\n", i);
if (cg840_select_board (i) != i)
{
printf ("Can not select board: %d\n", i);
return(-1);
}
/* Set the output gain to X1, X2, X5 or X10 */
switch (GAIN)
{
    case 1: cg840_set_gain(CG840_GAIN_X1); printf ("CompuGen 840A => Setting gain to %dX\n",GAIN); break;
    case 2: cg840_set_gain(CG840_GAIN_X2); printf ("CompuGen 840A => Setting gain to %dX\n",GAIN); break;
    case 5: cg840_set_gain(CG840_GAIN_X5); printf ("CompuGen 840A => Setting gain to %dX\n",GAIN); break;
    case 10: cg840_set_gain(CG840_GAIN_X10); printf ("CompuGen 840A => Setting gain to %dX\n",GAIN); break;
    default: cg840_set_gain(CG840_GAIN_X1); printf ("CompuGen 840A => Setting gain to %dX\n",GAIN);
exit(1);
}
if(SINGLE_SHOT)
{
printf ("CompuGen 840A => Setting to Single Shot.\n");
cg840_single_shot(TRUE);
}
/* Select Dual/Single Channel Mode (Analog & Digital) */
cg840_set_mode(MODE);

/* Set the CG840 Sample Conversion Clock Rate to 40 MHz */
/* Include various rates switching later */
printf ("\n");
switch (RATE)
{
    default: cg840_set_clock_rate(CG840_RATE_40, CG840_MHZ);
}
/* Dummy Stuff here. Fill in later. */
printf ("CompuGen 840A => Setting the sampling rate to 40 MHz output sampling rate.
");  
cg840_set_clock_rate (CG840_RATE_40, CG840_MHZ);  
cg840_need_ram (FALSE);  

/* Set triggering to Software Trigger and check for error.*/  
if (TRIGGER == SOFTWARE)  
{  
if (cg840_set_trigger_source (CG840_SOFTWARE_TRIG) != 
CG840_SOFTWARE_TRIG)  

printf("CompuGen 840A => Setting to SOFTWARE TRIGGERING\n");  
printf("CompuGen 840A => Error initializing trigger settings.\n");  
}  
if (TRIGGER == EXTERNAL)  
{  
if (cg840_set_trigger_source (CG840_EXTERNAL_TRIG) != 
CG840_SOFTWARE_TRIG)  

printf("CompuGen 840A => Setting to EXTERNAL TRIGGERING\n");  
printf("CompuGen 840A => Error initializing trigger settings.\n");  
}  

/* Turn on/off the filter */  
if (FILTER == FALSE)  
{  
printf ("CompuGen 840A => Turning off the filter.\n");  
cg840_set_filter_on (FALSE);  
}  
if (FILTER == TRUE)  
{  
printf ("CompuGen 840A => Turning on the filter.\n");  
cg840_set_filter_on (TRUE);  
}  

/* Set triggering to Software Trigger and check for error.*/  
if (cg840_set_trigger_source (CG840_SOFTWARE_TRIG) != 
CG840_SOFTWARE_TRIG)  

printf("CompuGen 840A => Error initializing trigger settings.\n");  
}  
if (cg840_select_board (1) != 1)  
{  
printf ("Can not reselect board: %d\n", 1);  
return(-1);  
}  
return(result);
double sqr(double x) {
    return(pow(x,2));
}

int convert_float_to_digital(double dpoint, double signal_max) {
    int result;
    result = 128 + (byte)(-dpoint/signal_max * 127);
    return(result);
}

float convert_digital_to_float(byte digital_val, double signal_max) {
    float result;
    result = signal_max/127*(128-digital_val);
    return(result);
}

int calculate_digital_trigger (cg840_buffer_type output_buffer) {
    int i, max;
    max = (1 << bits) - 1;
    printf("\nCompuGen 840A => Calculating digital trigger.\n");
    for (i = 0 ; i <= max ; i++) {
        output_buffer[CG840_FULL_DEPTH + i] = 255;
    }
    max = CG840_FULL_DEPTH - 1;
    for (i = CG840_DOUBLE_DEPTH + (1 << bits) ; i <= max ; i++) {
        output_buffer[i] = 0;
        return (1 << bits);
    }
}

int output_waveform (int DIGITAL_TRIG, cg840_buffer_type output_buffer) {
    int length;
}
/* Clear any current output of the CG840 */
cg840_abort ();

/* Here, we are loading the digital buffer. */
/* ------------------------------------------ */
if (DIGITAL_TRIG)
{
  if ((length = calculate_digital_trigger (output_buffer)) != (1 << bits))
  {
    return(-1);
  }
  if (cg840_load_buffer (2, length, output_buffer) != 2)
  {
    printf(" Error attempting to load Digital Buffer!
")
    printf(" Press a key to continue...");
    getch();
    return(-1);
  }
}

printf("CompuGen 840A => Preparing to dump data from buffer.\n");
printf("CompuGen 840A => Data dumped, waiting for trigger.\n");
return(0);

int calculate_sinc (int points, float time_start, float time_end,
                    float freq, int FILE_OUT, cg840_buffer_type
                    output_buffer)
{
  double t;
  int i;
  FILE *fp;

  printf("\nCompuGen 840A => Calculating Sine Pulse.\n");

  if (FILE_OUT)
  {
    printf("\nWrite Output File Selected\n");
    fp = fopen("SINC.TXT", "wb");
  }

  for (i = 0 ; i <= CG840_FULL_DEPTH-1 ; i++)
  {
    output_buffer[i] = 128;
  }

  for (i = 0 ; i <= points-1 ; i++)
  {
    t = ((double)(i) / (double)(points)*)((time_end-
    time_start))+(time_start);
    if (t!=0)
    {
      double t;
output_buffer[i] = convert_float_to_digital(-
1*sin(2*pi*freq*t)/(2*pi*freq*t), 1);
    if(FILE_OUT)
        fprintf(fp,"%g\r\n", -1*sin(2*pi*freq*t)/(2*pi*freq*t) );
    } else {
        output_buffer[i] = convert_float_to_digital(-1,1);
        if(FILE_OUT)
            fprintf(fp,"%g\r\n", (float) -1 );
    }
}
if(FILE_OUT)
    fclose(fp);
printf("\nCompuGen 840A => Trying to load CompuGen 840A buffer.\n");
/* Load DELAY times actual length of signal (just a experimental
   kludge for a 128 point signal)
Reason:
At end of Digital trigger, rather large negative pulse
comes on in the Analog line. By extending the length of
the analog signal, we 'push' the pulse far away so
we don't record it.
In the ACTUAL output, the Analog signal actually
occurs BEFORE the digital trigger.
Fortunately, there's delay in the system.
Careful, if you push the 'noise pulse' too far away, the
Analog signal will once again precede the trigger, even
through the system's delays. */
if (cg840_load_buffer (1, points*DELAY, output_buffer) != 1) {
    printf("ERROR attempting to load Analog Buffer!\n");
    return(-1);
} return (CG840_FULL_DEPTH);

int calculate_sin_t2 (int points, float time_start, float time_end,
            float freq, int FILE_OUT, cg840_buffer_type
            output_buffer)
{" double t;
    int i;
    FILE *fp;

    printf("\nCompuGen 840A => Calculating Sine Pulse.\n");
    if(FILE_OUT)
    { printf("\nWrite Output File Selected\n");
        fp = fopen("SIN_T2.TXT", "wb");
    }
    for (i = 0 ; i <= CG840_FULL_DEPTH-1 ; i++)
    { output_buffer[i] = 128;
    }
    for (i = 0 ; i <= points-1 ; i++)


```
{ 
    t = ((double)(i) / (double)(points)*(time_end-
        time_start))+(time_start);
    output_buffer[i] = convert_float_to_digital(-
        sin(2*pi*freq*t*t),1);
    if(FILE_OUT)
        fprintf(fp, "%g\r\n", -sin(2*pi*freq*t*t));
}
if(FILE_OUT)
    fclose(fp);
printf("\nCompuGen 840A => Trying to load CompuGen 840A buffer.\n"); /* Load DELAY times actual length of signal (just a experimental 
    kludge for a 128 point signal) 
    Reason: 
    At end of Digital trigger, rather large negative pulse comes on in the Analog line. By extending the length of 
    the analog signal, we 'push' the pulse far away so we don't record it. 
    In the ACTUAL output, the Analog signal actually occurs BEFORE the digital trigger. 
    Fortunately, there's delay in the system. 
    Careful, if you push the 'noise pulse' too far away, the 
    Analog signal will once again precede the trigger, even 
    through the system's delays. */
if (cg840_load_buffer (1, points*DELAY, output_buffer) != 1)
    { 
        printf("ERROR attempting to load Analog Buffer!\n");
        return(-1);
    }
return (CG840_FULL_DEPTH);
}
int calculate_sine (int points, float time_start, float time_end,
    float freq, int FILE_OUT, cg840_buffer_type
    output_buffer)
{ 
    double t;
    int i;
    FILE *fp;

    printf("\nCompuGen 840A => Calculating Sinc Pulse.\n");
    if(FILE_OUT)
    { 
        printf("\nWrite Output File Selected\n");
        fp = fopen("SINE.TXT", "wb");
    }

    for (i = 0 ; i <= CG840_FULL_DEPTH-1 ; i++)
    { 
        output_buffer[i] = 128;
    }
    for (i = 0 ; i <= points-1 ; i++)
    { 
        t = ((double)(i) / (double)(points)*(time_end-
            time_start))+(time_start);
```
output_buffer[i] = convert_float_to_digital(-
    sin(2*pi*freq*t), 1);
if (FILE_OUT)
    fprintf(fp, "%g\r\n", -sin(2*pi*freq*t));
}
if (FILE_OUT)
    fclose(fp);
printf("\nCompuGen 840A => Trying to load CompuGen 840A buffer.\n");
/* Load DELAY times actual length of signal (just a experimental
kludge for a 128 point signal)
Reason:
At end of Digital trigger, rather large negative pulse
comes on in the Analog line. By extending the length of
the analog signal, we 'push' the pulse far away so
we don't record it.
In the ACTUAL output, the Analog signal actually
occurs BEFORE the digital trigger.
Fortunately, there's delay in the system.
Careful, if you push the 'noise pulse' too far away, the
Analog signal will once again precede the trigger, even
through the system's delays. */
if (cg840_load_buffer (1, points*40, output_buffer) != 1)
{
    printf("ERROR attempting to load Analog Buffer!\n");
    return (-1);
}
return (CG840_FULL_DEPTH);

/****************************
Need to calculate the source pulse at each point in the array
(X,Y) specify the array element
c is the speed of sound (in water in this case)
spacing is the array element spacing (assuming square grid)
k, z0 are parameters specific to the FWM at a particular frequency
************************************************************
double calculate_fwm_point (float t, float spacing, int X, int Y,
float k, float z0)
{
    double si, s2, s3, s4, s5, s6, s7, s8, dpoint;
    double rhos, krho, zpct, zmct, kct;

    /************************************************************
    In the following section, we calculate the source pulse
required (at some specified location in the source array)
to generate a FWM pulse.
*************************************************************/
    rhos = sqrt(X*spacing)+sqrt(Y*spacing);
    zpct = sqrt(z0)+sqrt(c*t);
    zmct = sqrt(z0)-sqrt(c*t);
    kct = k*c*t;
    s1 = (double) (sqrt(zmct) + sqrt(2*z0*c*t));
    s2 = (double) sin(kct*rhos/zpct);
    s3 = (double) exp(-k*rhos*z0/zpct);
\[ s4 = (\text{double}) (z0\cos(kct) - c*t\sin(kct)) / -4/\pi/zpct; \]
\[ s5 = (\text{double}) \cos(kct*\text{rhos}/zpct); \]
\[ s6 = (\text{double}) (c*t\cos(kct) + z0*\sin(kct)) / 4/\pi/zpct; \]
\[ \text{dpoint} = (s6*s3*s5+s4*s3*s2) * (-2*kct*\text{rhos}*z0/s1+c*t/zpct) \]
\[ - (s4*s3*s5-s6*s3*s2) * (k*\text{rhos}*zmct/s1-z0/zpct+k); \]
\]
\[ \text{return}(\text{dpoint}); \]

```c
int calculate_fwm_pulse (int points, float time_range, float spacing, int X, int Y, float k, 
float z0, cg840_buffer_type output_buffer, 
double *Norm_Factor, int FILE_OUT)
{
    int icount, max;
    float t, temp_buffer[512];
    double dpoint;
    double signal_max;
    FILE *fp;

    /* Check to see if too many points were asked for */
    if (points > CG840_FULL_DEPTH)
        return (-1);

    /* Let's clear the buffer before we start */
    for (icount = 0 ; icount <= CG840_FULL_DEPTH-1 ; icount++)
        output_buffer[icount] = 128;

    if(FILE_OUT)
    {
        printf("\nWrite Output File Selected\n");
        fp = fopen("FWM.TXT", "wb");
    }

    max = points - 1;

    /******************************************************************************
     * This is a first pass so that we can find the maximum and normalize the signal. 
     * The signal will have to be 'un-normalized' before we process the data. 
     ******************************************************************************/
    signal_max = 0;
    printf("\nCompuGen 840A => Calculating normalizing factor.\n");
    for (icount = 0 ; icount <= max ; icount++)
    {
        /* Convert icount to a time value */
        t = ((double)(icount) / (double)(points)*time_range) - (time_range/2);
        dpoint = calculate_fwm_point(t, spacing, X, Y, k, z0);
        temp_buffer[icount]=dpoint;
        if(FILE_OUT)
            fprintf(fp, "%g\r\n", dpoint);
        if (fabs(dpoint) > signal_max)
```
In the following section, we calculate the source pulse required (at some specified location in the source array) to generate a FWM pulse. The final result, before writing to the CompuGen 840A buffer, must be normalized, due to the extremely high maximum value of the function.

```c
    t = ((double)(icount) / (double)(points)*time_range)-
(time_range/2);  
dpoint = calculate_fwm_point(t, spacing, abs(X), abs(Y), k,  
zO);  
    /*
     *--------End of pulse calculation-----------------------------*/
     
    output_buffer[icount] =  
convert_float_to_digital(temp_buffer[icount],signal_max);  
}
```

Now, convert values to byte format so it can be written properly to the CompuGen 840A buffer.

```
     *Norm_Factor = signal_max;
     
    /* Load DELAY times actual length of signal (just a experimental kludge for a 128 point signal) 
Reason:
    At end of Digital trigger, rather large negative pulse comes on in the Analog line. By extending the length of the analog signal, we 'push' the pulse far away so we don't record it. 
    In the ACTUAL output, the Analog signal actually occurs BEFORE the digital trigger. 
    Fortunately, there's delay in the system. 
    Careful, if you push the 'noise pulse' too far away, the Analog signal will once again precede the trigger, even through the system's delays. */
```
if (cg840_load_buffer (1, points*DELAY, output_buffer) != 1)
{
    printf("ERROR attempting to load Analog Buffer!\n");
    return(-1);
}
return (CG840_FULL_DEPTH);
} /* END */

control.c

/***************************************** *'******************************************
* Experiment_Control
* Version: 0.70
* Revision Date: February 6, 1996
* Written by: Isaac Leung
*
*********************************************************/

#include <math.h>
#include <conio.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cslitdrv.h"
#include "cg840drv.h"
#include "EXPMT.H"
#include "DELAYS.H"
#include "MS3_CTRL.H"

#define VERSION 0.70
#define CS220_DEFAULT_CONFIG_FILE "CS220.INC"
#define CG840_DEFAULT_CONFIG_FILE "CG840.INC"
#define DEFAULT_OUTPUT_DIRECTORY "C:/ISAAC/PROJECT/DATA/"

void main (int argc, char **argv)
{
    int choice;
    char
driver_840_file[255], driver_cslite_file[81], output_file[255];
    char
temp[16], answer;

c1rscr();

    /* Check command line for user supplied include file, if found use it.
     * otherwise try to use the files */
"CS220.INC" for the CompuScope LITE
"CG840.INC" for the CompuGen840A
"OUTPUT.TXT" for the output file

NOTE: Copy the .INC files to the working directory AFTER you have set up the boards with CSINST.EXE and CGINST.EXE */

if (argc == 1)
{
    strcpy (driver_cslite_file, CS220_DEFAULT_CONFIG_FILE);
    strcpy (driver_840_file, CG840_DEFAULT_CONFIG_FILE);
    strcpy (output_file, DEFAULT_OUTPUT_DIRECTORY);
} else
{
    if (argc == 4)
    {
        strcpy (driver_cslite_file, argv[1]);
        strcpy (driver_840_file, argv[2]);
        strcpy (output_file, argv[3]);
    } else
    {
        printf ("usage: FWM_EXP [CSLITE_driver_file]
[CG840A_driver_file] [Data_Directory]\n"),
        exit (1);
    }
}

/* Initialize the Gage boards. If everything okay, then proceed. */
if (EXP_init() == TRUE)
{
    printf("\n\n\nALRIGHT! Things seem to be working!\n"");
    clrscr();
    printf("******************************\n");
    printf("** Experiment Control V%3.2f */\n",VERSION);
    printf("******************************\n");
    printf("\n\n");
    printf("| Currently Available Experiments: |\n") ;
    printf("|--------------------------------\n") ;
    printf(" [0] Move the stage\n");
    printf(" [1] Generate and acquire single pulse\n");
    printf(" [2] Measure the amplitude decay with distance\n") ;
    printf(" [3] Synthetic FWM experiment\n");
    printf("\n Please enter Experiment Number: ");
    answer = getche();
    switch(answer)
    {
        case '3': EXP_synthetic_FWM(); break;
        case '2': EXP_decay_distance(); break;
        case '1': EXP_single_pulse(); break;
        case '0': EXP_move_stage(); break;
        default: EXP_single_pulse();
    exit(1);
    }
}
}  
else  
{  
    clrscr();  
    printf("There appears to have been an ERROR while attempting\n");  
    printf("to initialize the hardware.\n");  
}  
printf("Thank you for your support!\n");  
}  
/* End of main routine. */  

---

**cs_ctrl.h**

This is the header file for CS_CTRL.C  
A set of control routines for the  
CompuScope Lite Board  
***************************************/

#define BUF_SIZE 8192

int initialize_compuscope_lite (int MODE, int GAIN, int RATE,  
    int TRIGGER);

int cslite_grab_data(byte A_Buffer[BUF_SIZE+1],  
    byte B_Buffer[BUF_SIZE+1],int PRE_DATA);

void write_buffer_to_file(char *output_file, int DATA_FILE_LENGTH,  
    int WRITE_A, int WRITE_B, byte *A_Buffer,  
    byte *B_Buffer, double Norm_Factor);

/* END */

---

**cs_ctrl.c**

/**********************************************************  
* CompuScope LITE_Control  
* Version: 0.80  
* Revision Date: March 16, 1996  
* Written by: Isaac Leung  
*  
***********************************************************/

#include <conio.h>
#include <process.h>
#include <stdio.h>
#include <math.h>
#include "cslitdrv.h"
#include "cg840drv.h"
#include "CS_CTRL.H"
#include "DELAYS.H"
#define VERSION 0.8
#define BUF_SIZE 8192
#define CS220_DEFAULT_CONFIG_FILE "CS220.INC"
#define EXTERNAL CSLITE_EXTERNAL

int initialize_compuscope_lite (int MODE, int GAIN,
           int RATE, int TRIGGER)
{
    int i, result_cslite, result, count;
    long trigger_address, address;
    byte *cslite_board_location;

    result = FALSE;
    clrscr();
    printf("******************************
");
    printf("* Initialize CompuScope Lite V%3.2f *
", VERSION);
    printf("******************************
");
    cslite_read_config_file
    (CS220_DEFAULT_CONFIG_FILE,cslite_board_location);
    result_cslite = cslite_driver_initialize (cslite_board_location,
            CSLITE_MEMORY_SIZE_TEST);
    if (result_cslite == 0)
    {
        printf(" No CompuScope LITE boards were found.\n");
        printf(" An error occurred!\n");
        printf(" Press a key to end...\n");
        getch();
        exit (1);
    }
    else
    {
        if (result_cslite < 0)
        {
            printf(" Not all CompuScope LITE boards were
found.\n");
            printf(" Number of CompuScope LITE boards found =
%d\n", -result_cslite);
            printf(" An error occurred!\n");
            printf(" Press a key to end...\n");
            getch();
            exit (1);
        }
        /* Okay everything seems to be okay with the CSLITE */
    }
    for (i = 1 ; i <= result_cslite ; i++)
if (cslite_select_board (i) != i)
{
    char str [81];
sprintf (str, "An error was encountered selecting board 
%d.", i);
    printf (str);
    printf ("press any key to continue.");
    getch();
    continue;
}

/* Set Single/Dual Channel Mode */
/* Leave Rate as a dummy setting for now */
switch (RATE)
{
    /* Dummy stub for now */
    default: cslite_capture_mode(MODE, CSLITE_RATE_20,
    CSLITE_MHZ);
    cslite_capture_mode (MODE, CSLITE_RATE_20, CSLITE_MHZ);
    if(GAIN == 1)
    {
        /* Dummy Stub for now */
        if(GAIN == 5)
        {
            /* Dummy Stub for now */
            cslite_input_control (CSLITE_CHAN_A, CSLITE_INPUT_ENABLE,
            CSLITE_AC, CSLITE_TIMES_5);
cslite_input_control (CSLITE_CHAN_B, CSLITE_INPUT_ENABLE,
            CSLITE_AC, CSLITE_TIMES_1);
            if(TRIGGER == EXTERNAL)
            {
                /* Dummy Stub for now */
                if(cslite_trigger_control (CSLITE_EXTERNAL, CSLITE_AC,
                CSLITE_TIMES_1, CSLITE_POSITIVE, 0x90, CSLITE_POST_8K)==0)
                {
                    printf("CompuScope Lite=> ERROR in setting trigger
parameters!\n");
                    printf("Press a key to continue...");
                    getch();
                }
            /* Gotta allow time for the board to set up
else, you get a bunch of noise at the
beginning of the signal. */
            printf("\n");
            printf("CompuScope Lite=> Waiting for board to initialize.");
        }
        return(result);
    }
# Appendices

```c
int cslite_grab_data(byte A_Buffer[BUF_SIZE+1], byte B_Buffer[BUF_SIZE+1], int PRE_DATA)
{
    int count, result;
    long address;

    result = 0;

    printf("CompuScope Lite=> Triggering CompuGen 840A.\n");
    printf("CompuScope Lite=> Enabling Channel A.\n");
    cslite_channel_enable(CSLITE_CHAN_A, CSLITE_INPUT_ENABLE);
    printf("CompuScope Lite=> Enabling Channel B.\n");
    cslite_channel_enable(CSLITE_CHAN_B, CSLITE_INPUT_ENABLE);
    printf("CompuScope Lite=> Getting data into CompuScope LITE RAM.\n");

    printf("Now Dumping CompuGen 840A Data\n");
    cg840_dump_data();
    printf("Getting data\n");
    cslite_get_data();
    printf("Calling CompuGen 840A Trigger\n");
    cg840_software_trigger();

    printf("CompuScope Lite=> Awaiting Trigger...\n");
    while(!cslite_triggered())
    {
        /* Wait till CSLITE is triggered */
    }
    while(cslite_busy())
    {
        /* Hold on if CSLITE is busy */
    }

    printf("CompuScope Lite=> Reading CompuScope RAM into buffer.\n");
    address = cslite_trigger_address()-PRE_DATA;
    cslite_need_ram(TRUE);
    count = 0;

    while(count <= BUF_SIZE)
    {
        A_Buffer[count]=cslite_mem_read_chan_a(address);
        B_Buffer[count]=cslite_mem_read_chan_b(address);
        address++;
        count++;
    }
    cslite_need_ram(FALSE);

    return(result);
}
```

```c
void write_buffer_to_file(char *output_file, int DATA_FILE_LENGTH,
                          int WRITE_A, int WRITE_B, byte *A_Buffer,
                          byte *B_Buffer, double Norm_Factor)
{
    int count;
    FILE *fp;
```
fp = fopen(output_file, "wb");
printf("CompuScope Lite=> Output File: %s\n", output_file);
count = 0;
printf("CompuScope Lite=> Writing buffer to File.\n");

if(Norm_Factor != 1)
{
    fprintf(fp,"%g", Norm_Factor);
    if(WRITE_A && WRITE_B)
        fprintf(fp," %g", Norm_Factor);
    fprintf(fp, "\r\n");
}

while(count < DATA_FILE_LENGTH)
{
    if(WRITE_A)
        fprintf(fp,"%f6",(float)(128-A_Buffer[count])/127);
    fprintf(fp," ");
    if(WRITE_B)
        fprintf(fp,"%f6",(float)(128-B_Buffer[count])/127);
    fprintf(fp, "\r\n");
    count++;
}
fclose(fp);

/ * END */

delays.h

/****************************
Header file for DELAYS.C
A set of utility routines
for delays and alerts.
****************************/

int wait(int Delay_Length);
int wait2(int Delay_Length);
int wait3(void);
void alertl(int length);

/ * END */

delays.c

/ *****************************/

DELAYS.C

Author: Isaac Leung
Date: March 12, 1996
Description:
This is a set of utility routines used to generate various
delay functions, alerts, etc.

******************************************************************************

#include <conio.h>
#include <time.h>
#include <stdio.h>
#include <dos.h>

int POSITION=0;

int wait3()
{
    time_t t1;
    char spin[4];
    int oldx,oldy;

    spin[0] = 0xC4; /* - */
    spin[1] = 0x5C; /* \ */
    spin[2] = 0xB3; /* | */
    spin[3] = 0x2F; /* / */

    oldx=wherex();
    oldy=wherey();
    gotoxy(79,2);
    textcolor(10);
    cprintf("%c",spin[POSITION]);
    textcolor(16);
    delay(8);

    if(POSITION == 3)
        POSITION = 0;
    else
        POSITION = POSITION + 1;
    gotoxy(oldx,oldy);
    return(0);
}

int wait2(int Delay_Length)
{
    time_t t1;
    char spin[4];
    int count,oldx,oldy;

    spin[0] = 0xC4; /* - */
    spin[1] = 0x5C; /* \ */
    spin[2] = 0xB3; /* | */
    spin[3] = 0x2F; /* / */

    oldx=wherex();
    oldy=wherey();
    t1 = time(NULL);
    while((time(NULL) - t1) <= Delay_Length)
    {
        count = 0;
        /* Delay code */
    }
    return(0);
}
while (count <=3)
{
    gotoxy(80,1);
textcolor(10);
cprintf("%c",spin[count]);
textcolor(16);
delay(64);

    count++;
}
}
gotoxy(oldx,oldy);
textcolor(7);
printf("\n");

return(0);

int wait(int Delay_Length)
{
    time_t t1;
    char spin[8],ball;
    int count,oldx,oldy;
    int x[8],y[8];

    spin[0] = 0xC4; /* - */
    spin[1] = 0x5C; /* \ */
    spin[2] = 0xB3; /* | */
    spin[3] = 0x2F; /* / */
    spin[4] = 0x5C;
    spin[5] = 0xB3;
    spin[6] = 0x2F;
    spin[7] = 0xC4;

    ball = 0xF9;

    x[0] = 80; y[0] = 2;
    x[1] = 80; y[1] = 3;

    oldx=wherex();
    oldy=wherey();
t1 = time(NULL);
while((time(NULL) - t1) <= Delay_Length)
{
    count = 0;
    while (count <=7)
    {
        if(count == 0)
        {
            gotoxy(x[7],y[7]);
        }
    

```
else
{
    gotoxy(x[count-1],y[count-1]);
}
cprintf("*");
goxy(x[count],y[count]);
textcolor(14);
cprintf("%c",ball);
goxy(79,2);
textcolor(10);
cprintf("%c",spin[count]);
textcolor(16);
delay(16);

count++;
}
goxy(oldx,oldy);
printf("\n");
textcolor(7);
return(0);
}

void alertl(int length)
{
    int count;
    for(count = 0 ; count < length ; count++)
        printf("%c",0x07);
}

/ * END */

expmt.h

******************************************************************************
This is the header file for EXPMT.C
A set of routines for controlling various experiments.
******************************************************************************

int EXP_init(void);
void EXP_move_stage(void);
void EXP_single_pulse(void);
void EXP_decay_distance(void);
void EXP_synthetic_FWM(void);

/ * END */

expmt.c

******************************************************************************
* Experiment_List *
******************************************************************************

M.A.Sc. Thesis Isaac Jing Herng Leung 06/07/96
int EXP_init()
{
    int success, OK_CSLITE, OK_840, SINGLE_SHOT, OUTPUT_GAIN, temp;
    char answer;

    SINGLE_SHOT = TRUE;
    OK_840 = FALSE;
    OK_CSLITE = FALSE;
    success = FALSE;
    OUTPUT_GAIN = 0;

    clrscr();

    /* Initialize the CompuScope LITE driver and report the number of boards
       found. If all is correct then continue with the program. */

    printf("*****************************************
    printf("** Experiment Equipment InitializationV%3.2f */
    printf("*****************************************
    printf("\n");
    printf("=> Attempting to Initialize CompuScope Lite board.\n");
    printf(" Status:\n");
    printf("-------------------------------------\n");
    OK_CSLITE = initialize_compuscope_lite (CSLITE_DUAL_CHAN, 1,20,
    CSLITEEXTERNAL);
    /* Initialize the CompuGen 840A driver and report the number of boards
       found. If all is correct then continue with the program. */

    printf("\n");
    printf("=> Attempting to Initialize CompuGen 840A board.\n");
    printf(" Status:\n");
    printf("-------------------------------------\n");
/* For now, we arbitrarily set some values for the boards to work with.
*/

printf("\n");
printf("Compugen 840A Output Level +/-[ 1 | 2 | 5 | 10]: ");
scanf("%d", &OUTPUT_GAIN);
printf("Do you want Single Shot output (y/n)? ");
answer=getche();
if ((answer == 'n') || (answer == 'N'))
{
    SINGLE_SHOT=FALSE;
}

OK_840 = initialize_cg840 (CG840_DUAL_MODE, SINGLE_SHOT, OUTPUT_GAIN, 20, CG840_SOFTWARE_TRIG, FALSE);
if ((OK_840 == TRUE) && (OK_CSLITE == TRUE))
success = TRUE;
return(success);

void EXP_move_stage(void)
{
    float range;
    char answer;
    clrscr();
    printf("##########################
");
    printf("# Experiment: Move Stage #\n");
    printf("##########################\n");
    printf("Please enter all values in meters\n");
    scanf("%g", &range);
    move_stage_straight(range, METERS,0);
    printf("Shall I move the stage back to the start position (y/n)? ");
    answer = getche();
    if ((answer == 'y') || (answer == 'Y'))
    {
        move_stage_straight(-range, METERS,0);
    }
}

void EXP_single_pulse()
{
    byte A_Buffer[8193], B_Buffer[8193],
    output_buffer[CG840_DOUBLE_DEPTH];
    double *Norm_Factor;
    float range, tstart, tend, freq, TA_Buffer[1025], TB_Buffer[1025];
    int choice, X,Y,points,N,count,count2,N_out,PRE_DATA;
    char fname[16],answer;
range = 0;

/* Note: These buffers are for temporary collection and averaging.
   Unfortunately, due to memory constraints, if this 'option'
   is compiled in, the maximum number of points we can
   collect is 1024.
*/

for(count2 = 0 ; count2 <= 1025 ; count2++)
{
    TA_Buffer[count2] = 0;
    TB_Buffer[count2] = 0;
}

clrscr();
printf("########################################
");
printf("# Experiment: Single Pulse #
");
printf("########################################
");
printf(" 

Please enter all values in meters 

Output filename: ");
scanf("%s", &fname);
printf("Measuring distance: ");
scanf("%g", &range);
printf("Number of points to collect: ");
scanf("%d", &points);
printf("Number of runs to average: ");
scanf("%d", &N);

printf("[1] Pseudo-Impulse
");
printf("[2] Sinc Pulse
");
printf("[3] FWM Pulse
");
printf("[4] sin(t)
");
printf("[5] sin(t*t)
");
printf("Type of pulse: ");
choice=getche();

if(choice == '1')
{
    tstart=-3.2e-6;
    tend=3.2e-6;
    calculate_sinc (128, tstart, tend, 9.5E6, TRUE, output_buffer);
/* Okay, PRE_DATA is pretty kludgy. It's different for different distances
   for 12cm, use - 930;
   for 6cm, use -1760;
direct D/A -> A/D, use 2600*/
    PRE_DATA=460;
}
if(choice == '2')
{
    tstart=-6.4e-6;
    tend=6.4e-6;
    printf("Frequency[9E6]: ");
    scanf("%g", &freq);
    calculate_sinc(256,tstart,tend,freq,TRUE,output_buffer);
    PRE_DATA=2600;
}
if(choice == '3')
{
printf("Which source pulse [X,Y]?\n");
printf("X: ");
scanf("%d", &X);
printf("Y: ");
scanf("%d", &Y);
calculate_fwm_pulse (256, 1.28E-5, 0.0018, X, Y, 5, 4E-4, output_buffer,Norm_Factor, TRUE);
PRE_DATA=930;
printf("PreDATA= ");
scanf("%d",&PRE_DATA);
}
if(choice == '4')
{
printf("Frequency[1.0E6]: ");
scanf("%g", &freq);
tstart=0;
/* Actual Freq=freq*total_time/128samples*20E6Samples/sec. */
tend=l;
N_out=(int) (20E6/freq);
calculate_sine(N_out,tstart,tend,1,TRUE,output_buffer);
PRE_DATA=500;
}
if(choice == '5')
{
printf("Start time [0]: ");
scanf("%g", &tstart);
printf("End Time [1.6E-4]: ");
scanf("%g", &tend);
calculate_sin_t2(128,tstart,tend,5E8,TRUE,output_buffer);
PRE_DATA=10;
}
move_stage_straight(range, METERS,0);
calculate_digital_trigger(output_buffer);
output_waveform (TRUE, output_buffer);
wait(1);
/* Note: cslite_grab_data also initiates the waveform output */
count = 1;
while(count <= N)
{
cslite_grab_data(A_Buffer, B_Buffer,PRE_DATA);
for(count2=0 ; count2 <= 1025 ; count2++)
{
TA_Buffer[count2]=TA_Buffer[count2]+convert_digital_to_float(A_Buffer[count2],1);
TB_Buffer[count2]=TB_Buffer[count2]+convert_digital_to_float(B_Buffer[count2],1);
}
for(count2=0 ; count2 <= 1025 ; count2++)
{
A_Buffer[count2]=convert_float_to_digital(TA_Buffer[count2]/N,1);
B_Buffer[count2]=convert_float_to_digital(TB_Buffer[count2]/N,1);
}
write_buffer_to_file(fname,points, TRUE, TRUE, A_Buffer, B_Buffer,1);
printf("Shall I move the stage back to the start position (y/n)? ");
answer = getche();
if ((answer == 'y') || (answer == 'Y'))
{
    move_stage_straight(-range, METERS,0);
}

void EXP_decay_distance()
{
    byte A_Buffer[8193], B_Buffer[8193],
    output_buffer[CG840_DOUBLE_DEPTH], peaks_buffer[1024];
    double *Norm_Factor;
    float range, min, tstart, tend, freq, max;
    int count, count2, count3, N, index, num_points, choice, GAIN, N_out, X, Y, PRE_DATA
    char fname[16], answer;

    range = 0;
    for(count2 = 1 ; count2 <= 1024 ; count2++)
    {
        peaks_buffer[count2] = 0;
    }

    clrscr();
    printf("# Experiment: Amplitude Decay vs. Distance #
    printf("Please enter all values in meters\n");
    printf("Number of data points: ");
    scanf("%d", &num_points);
    printf("[1] 1 x Receiver Gain = +/- 1.0 V (Channel B)\n");
    printf("[5] 5 x Receiver Gain = +/- 1 200 mV (Channel A)\n");
    printf("Type of pulse: ");
    choice = getche();
    printf("\n");
    if(choice == '1')
    {
        printf("Start time [-1.0E-6]: ");
        scanf("%g", &tstart);
        printf("End Time [1.0E-6]: ");
        scanf("%g", &tend);
    }
}
scanf("%g", &tend);
printf("Frequency[2E7]: ");
scanf("%g", &freq);
calculate_sinc(128,tstart,tend,freq,TRUE,output_buffer);
PRE_DATA=10;
}
if(choice == '2')
{
 printf("\nWhich source pulse [X,Y]?\n");
 printf("X: ");
 scanf("%d", &X);
 printf("Y: ");
 scanf("%d", &Y);
calculate_fwm_pulse (128, 1.4E-6, 0.00092,X, Y, 12, 0.6E-4,
 output_buffer, Norm_Factor, TRUE);
PRE_DATA=1280;
}
if(choice == '3')
{
 printf("Frequency[1.0E6]: ");
 scanf("%g", &freq);
tstart=0;
/* Actual Freq=freq*total_time/128samples*20E6Samples/sec. */
tend=1;
N_out=(int) (20E6/freq);
calculate_sine(N_out,tstart,tend,1,TRUE,output_buffer);
PRE_DATA=100;
}
calculate_digital_trigger(output_buffer);

index=0;
while(index < num_points)
{
 move_stage_straight(range/num_points, METERS,0);
 output_waveform (TRUE, output_buffer);
 wait(1);
/* Note: cslite_grab_data also initiates the waveform output */
/* Start Averaging Loop */
count2 = 1;
while(count2 <= N)
{
 cslite_grab_data(A_Buffer, B_Buffer,PRE_DATA);
count = 0;
/* Use min because in byte form, 0 is +1 Volts, 255 is -1 Volts */
/* Use min or max to detect either peaks or valleys */
 min=255;
 max=0;
 for (count = 0 ; count <= 4096 ; count++)
 {
 if(GAIN == 1)
 {
 if(B_Buffer[count] < min)
 min=B_Buffer[count];
 }
 if(GAIN == 5)
{ 
    if(A_Buffer[count] < min)
        min=A_Buffer[count];
}

peaks_buffer[index]=(peaks_buffer[index]*(count2-1)+min)/count2;
    count2++;
} /* End Averaging Loop */
index++;
}

write_buffer_to_file(fname,num_points, TRUE, FALSE, peaks_buffer, B_Buffer, 1);
    alert1(32);
    printf(" Shall I move the stage back to the start position (y/n)? ");
    answer = getche();
    if ((answer == 'y') || (answer == 'Y'))
{
    move_stage_straight(convert_distance_to_steps(range/num_points)*num_points, STEPS,0);
}
}

void EXP_synthetic_FWM()
{
byte A_Buffer[8193], B_Buffer[8193],
output_buffer[CG840_DOUBLE_DEPTH];
double *Norm_Factor,distance,current_distance;
float range,radius,k,z0,Source_Spacing,radii[16];
t
int count,X_MAX,Y_MAX,GAIN,i,it,j,jt,PRE_DATA,N,count2,rcount,cstep;
char output_dir[128],temp[16],output_file[255],answer;

range = 0.12;
radius = 0.015;
cstep=0;
strcpy(output_file,DEFAULT_OUTPUT_DIRECTORY);

count = 0;
while(count <= 15)
{
    radii[count] = radius/15*count;
    count++;
}

current_distance = 0;
Source_Spacing = 0.0018;
k = 5;
z0 = 4E-4;
X_MAX = 10;
Y_MAX = 10;
GAIN = 5;
PRE_DATA=930;
clrscr();
printf("########################################\n");
printf("# Experiment: Synthetic FWM Generation #\n");
printf("########################################\n");
printf("\n\n");

printf("Output directory [%s]: ",DEFAULT_OUTPUT_DIRECTORY);
scanf("%s",output_dir);
printf("\n Do you wish to enter your own data values (y/n)? ");
answer=getche();
if ((answer == 'y') || (answer == 'Y'))
{
  printf("\n\n") ;
  printf("Please enter all distances in meters. \n\n") ;
  printf("Enter the value for 'k' [%g]: ",k);
  scanf("%g", &k);
  printf("Enter the value for 'z0' [%g]: ",z0);
  scanf("%g", &z0);
  printf("\n") ;
  printf("Enter the Source Spacing [%6.5f]: ",Source_Spacing);
  scanf("%g", &Source_Spacing);
  printf("Enter the central-detector to central-source distance [%4.3f]: ",range);
  scanf("%g", &range);
  printf("Maximum radius out from central detector [%4.3f]?\n",radius);
  scanf("%g", &radius);
  count = 0;
  while(count <= 15)
  {
    radii[count] = radius/15*count;
    count++;
  }
  printf("\n=> Source array is [-X,X] wide and [-Y,Y] tall.\n");
  printf(" Enter value for X: ");
  scanf("%d", &X_MAX);
  printf(" Enter value for Y: ");
  scanf("%d", &Y_MAX);
  printf("\n") ;
  printf("Enter the value for output gain <1>,<2>,<5> or <10>: ");
  scanf("%d", &GAIN);
}

printf("\nThe experiment will proceed with the following values: \n");
printf("\n") ;
printf(" Value for 'k' = %g\n", k);
printf(" Value for 'z0' = %g\n", z0);
printf(" Source Spacing = %g\n", Source_Spacing);
printf(" Central source to central detector = %g m\n", range);
printf(" Radius out from central detector = %g m\n", radius);
printf(" Source elements = [-%d,%d] in width\n",X_MAX, X_MAX);
printf(" Source elements = [-%d,%d] in height\n",Y_MAX, Y_MAX);
printf("\n");
printf(" Output Gain = %d X\n",GAIN);
wait(1);
rcount = 0;
while(rcount <= 15)
{ 
  i = -X_MAX;
  while (i <= X_MAX)
  {
    j = 0;
    while (j <= Y_MAX)
    {
      /* Here, we're just building up the output file name. */
      itoa(rcount,temp, 10);
      strcat(output_file, temp);
      strcat(output_file, "/");
      strcat(output_file, "x");
      if (i <0 )
        strcat(output_file, ";");
      it=abs(i);
      itoa(it,temp, 10);
      strcat(output_file, temp);
      strcat(output_file, "/y");
      if (j < 0)
        strcat(output_file, ";");
      jt=abs(j);
      itoa(jt,temp, 10);
      strcat(output_file, temp);
      strcat(output_file, ".TXT");
      printf("Output File: %s\n",output_file);
    /* End build output file name. */
    
      distance = sqrt( sqr(Source_Spacing*j)+sqr(radii[rcount]-
                     Source_Spacing*i)+sqr(range));
      cstep=move_stage_straight (-distance, METERS,cstep);
      current_distance=distance;
      
      calculate_fwm_pulse (256, 1.28E-5, Source_Spacing,i, j,
                           k,z0, output_buffer,Norm_Factor,FALSE);
      calculate_digital_trigger(output_buffer);
      output_waveform (TRUE, output_buffer);
      wait(1);
      
      cslite_grab_data(A_Buffer, B_Buffer,PRE_DATA);
      write_buffer_to_file(output_file,512, FALSE, TRUE,
                           A_Buffer, B_Buffer, *Norm_Factor);
    /* Restore the output file name to just the directory
      name */
      strcpy (output_file, DEFAULT_OUTPUT_DIRECTORY);
    j++;
  } /* End of looping through Y */
  i++;
} /* End of looping through X */
rcount++;
} /* End of looping through the radii */
} /* END */
ms3_ctrl.h

/****************************
This is the header file for MS3_CTRL.C 
A set of routines for controlling the 
MSTEP-3 Stepper Motor Driver. 
****************************/

#define STEPS 0
#define METERS 1

int move_stage_straight(double distance, int UNITS, int csteps);
int convert_distance_to_steps(double distance);
/* END */

ms3_ctrl.c

/***************************************************************************/
/* MSTEP-3_Control */
/* Version: 1.05 */
/* Revision Date: April 16, 1996 */
/* Written by: Isaac Leung */
/***************************************************************************/
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <process.h>
#include "MS3_CTRL.H"
#include "DELAYS.H"
#define VERSION 1.02
#define Degrees_per_Step 1.8
#define Meters_per_Revolution 0.00508
#define STEPS 0
#define METERS 1
#define Motion_Exec "MOVE6.EXE"
#define Command_File "MOTION.CMD"

int move_stage_straight(double distance, int UNITS, int csteps) 
{
    int steps,speed,t;
    FILE *fp;
    clrscr();
    printf("************************
");
    printf("* FWM Move It! V%3.2f *
",VERSION);
    printf("* Module: Straight Move *
");
    printf("************************
");
    if(UNITS != STEPS)
steps = convert_distance_to_steps(distance);
else
    steps = distance;
printf("MSTEP-3 => Distance to move = %g m\n",distance);
printf(" Half Steps = %d\n",steps);

fp = fopen(Command_File, "wb");
printf("MSTEP-3 => Writing to command file.\n");
/* Dummy variable speed, it's not actually required for this implementation of move relative */
speed = 40;
fprintf(fp, "%d %d %d\r\n",3,steps-csteps,speed);
fclose(fp);
printf("MSTEP-3 => Initiate Move Relative.\n");
if(steps != 0)
{
    system(Motion_Exec);
}

/* Okay, we need a delay loop to allow the Motion stage to get into position. We ought to be able to query it to see if it's busy, but, it's in QuickBASIC only, and it seems to generate error if you query it when it's busy!! */
printf("MSTEP-3 => Waiting for motion stage to get into position.\n");
if ((UNITS != STEPS) && (csteps == 0))
    t = abs((int) ceil(2*distance/0.03))+1;
else
    t = abs(ceil(2 *distance/2362))+1;
wait(t);
return(steps);

int convert_distance_to_steps(double distance)
{
    int msteps;
    double asteps;

    printf("MSTEP-3 => Converting distance to steps.\n");

    /* The 2 comes from half stepping
    The 360 comes from 360 degrees per revolution */
    asteps = (double)(distance/Degrees_per_Step/Meters_per_Revolution*2.0*360.0);
    if((asteps-((int)asteps)) >= 0.5)
        msteps=(int)ceil(asteps);
    else
        msteps=(int)floor(asteps);

    return(msteps);
}

/* END */
m_create.c

 /******************************************************************************
 M_CREATE.C

 Author: Isaac Leung
 Date: April 23, 1996

 Description:
 This is a C program used to generate the Matlab program which
 is in turn used to process the experiment data in order to reconstruct
 a FWM pulse.
 Matlab is not very good at handling strings and also apparently
 does not allow loading within loops. Therefore, a program to create
 this file is required.
 For a 21x21 source array and 16 sets of radii measurement, the
 resulting Matlab file is on the order of 800kb in size.
 ******************************************************************************/

 #include <stdio.h>
 #include <stdlib.h>

#define Output_File "ADD_FWM.M"
#define Num_Radii 16
#define Default_File "." /* This is the default directory */

void main(void)
{
    FILE *fp;
    char *Data_File,*Disk_File,temp[16];
    int X_MAX,
    Y_MAX,i,j,t,radii_count,pend,pstart,Num_Points,length,N;
    float DC_OFFSET;

    strcpy(Disk_File, "");
    strcpy(Data_File, "");

    clrscr();
    N=12;
    printf("Enter the value for X_MAX: ");
    scanf("%d", &X_MAX);
    printf("Enter the value for Y_MAX: ");
    scanf("%d", &Y_MAX);
    /* Where is the start of the 'useful' data? */
    printf("Extract data START POINT: ");
    scanf("%d", &pstart);
    /* How many data points are 'useful'? */
    printf("Number of Data Points to collect: ");
    scanf("%d", &length);
    pend=pstart+length-1;
    Num_Points=pend-pstart+1;
    printf("Creating MATLAB file.");
    fp = fopen(Output_File, "wb");


M.A.Sc. Thesis Isaac Jing Herng Leung 06/07/96
fprintf(fp, "% This is a MATLAB file for processing my data
files.\n");
fprintf(fp, "\n"");
fprintf(fp, "pstart=%d;\n",pstart);
fprintf(fp, "pend=%d;\n",pend);
fprintf(fp, "SN=%d;\n",Num_Points);
fprintf(fp, "ST=SN/20E6;\n");
fprintf(fp, "SigTime=0:ST/(SN-1):ST;\n");
fprintf(fp, "SigFreq=zeros(1,SN);\n");
fprintf(fp, "for j = 1:1:SN\n");
fprintf(fp, " SigFreq(j) = j/ST;\n") ;
fprintf(fp, "end\n");
fprintf(fp, "FWM_Data = zeros(%d,%d);\n",Num_Points,Num_Radii);
/* fprintf(fp,"load %s.mat\n",Transfer_Function);
fprintf(fp,"load %s.mat\n",Weiner_Filter);
fprintf(fp,"load %s.mat\n",Low_Pass_Filter); */
fprintf (fp, "hpf=firl (256, 3E5/10E6, 'high' , bartlett (257) );\n") ;
radii_count = 0;
while(radii_count < Num_Radii)
{
 fprintf(fp, "ssdisp(0,'\%d of \%d done')\n",radii_count+1,Num_Radii);
i = -X_MAX;
while (i <= X_MAX)
{
 j = 0;
while (j <= Y_MAX)
{
 printf("."");
 /* Here, we're just building up the file name. */
 strcat (Data_File, "x");
 if (i < 0)
 strcat (Data_File, ":");
 it=abs(i);
 itoa(it,temp, 10);
 strcat (Data_File, temp);
 strcat (Data_File, ":y");
 if (j < 0)
 strcat (Data_File, ":");
 jt=abs(j);
 itoa(jt,temp, 10);
 strcat (Data_File, temp);
 /* End Build Data File Name */

 /* Build the actual Disk File Name */
 strcat(Disk_File, Default_File);
 strcat(Disk_File, "\") ;
 itoa(radii_count,temp,10);
 strcat(Disk_File,temp);
 strcat(Disk_File, "\") ;
 strcat(Disk_File, Data_File);
 strcat (Disk_File, ".txt");
 /* End build Disk File name. */

 /* Okay do what you need to do for EACH data file */
fprintf(fp,"load %s\n",Disk_File);

/*
fprintf(fp,"ssdisp(0,'%s')\n",Disk_File);
fprintf(fp,"Norm_Factor = %s(1)\n",Data_File);
fprintf(fp,"Current_Data = (%s(%d:%d)'-
mean(%s(%d:%d)) ));\n",Data_File,pstart,pend,Data_File,pstart,pend);

/* This is for the High pass filter method */
fprintf(fp,"yf=conv(hpf,Current_Data-
mean(Current_Data));\n")
fprintf(fp,"yfe=yf(129:129+255);\n")
fprintf(fp,"yi=cumsum(yfe);\n")
fprintf(fp,"Corrected_Data=cumsum(yi-
mean(yi));\n")

/* End HP Filter Method */

/* This is for the curve fit method */
fprintf(fp,"yi=cumsum(Current_Data);\n")
fprintf(fp,"f0=polyfit(SigTime,yi,%d);\n",N)
fprintf(fp,"Corrected_Data=cumsum(yi-
polyval(f0,SigTime));\n")

/* End Curve Fit Method */

/*
fprintf(fp,"Corrected_Data = real(ifft(
fft(Current_Data).*%s.*%s));\n",\n" Weiner_Filter,Low_Pass_Filter);*/

/* Except for the ONE set of sources on the y-axis, there are
TWO symmetric sources above and below the measurement axis */
if (j == 0)
{
   fprintf(fp,"FWM_Data(:,%d) = FWM_Data(:,%d) +
1*Norm_Factor*Corrected_Data'/l;\n",radii_count+l,radii_count+l);
}
else
{
   fprintf(fp,"FWM_Data(:,%d) = FWM_Data(:,%d) +
2*Norm_Factor*Corrected_Data'/l;\n",radii_count+l,radii_count+l);
}

/* Clear the variable to save memory */
fprintf(fp,"clear %s\n",Data_File);

/* End of what you need to do to each data file */

/* Reset the File Names */
strcpy (Data_File, "");
strcpy (Disk_File, "");

j++;

} /* End of Y_MAX loop */

i++;

} /* End of X_MAX loop */
radii_count++;
}

fclose(fp);

printf("\n*** Done ***\n");
} /* END */
B.1.2 Maple Program Files

This section contains the Maple program files used to perform some calculations on localized waves. Maple V R3 for both UNIX and Windows were used.

Maple V program for describing the FWM acoustical wave.

Date: March 31, 1996 (12:03 a.m.)

> with(plots);

At 0.6 MHz, use $k=2$, $Z[0]=4.5E-4$, spacing=0.003 At 1.0 MHz, use $k=20$, $Z[0]=2E-4$, spacing=0.00005

At 3.5 MHz, use $k=12$, $Z[0]=6E-3$, spacing=0.00046 At 10 MHz, use $k=20$, $Z[0]=2E-5$, spacing=0.00001

> c := 1540; k := 12; Z[0] := 0.6E-4; delta[X] := 0.00092/2; delta[Y] := 0.00092/2;

$c := 1540$

$k := 12$

$Z[0] := 0.6E-4$

$\Delta X := 0.00092/2$

$\Delta Y := 0.00092/2$

> rect := (start, pulse_width, t) -> (Heaviside(t-start)-Heaviside(t-start-pulse_width));

rect :=

$(start, pulse_width, t) \rightarrow Heaviside(t—start) - Heaviside(t—start-pulse_width)$

This is the equation describing the FWM, use $c$, $k$, and $Z[0]$

$\psi[FWM] := (x, y, z, t) \rightarrow \frac{1}{4} e^{i k (x + c t) \rho^2 / Z_0 + I (z - c t)} \int e^{i k (x + c t) \rho^2 / Z_0 + I (z - c t)}$

Well, this is the source signal, which should be $d\psi[FWM] + d\psi[S]$

$\psi[FWM] := (0, \sqrt{x^2 + y^2}), t) \left( \frac{I k (x^2 + y^2)}{(Z_0 + I (z - c t))} - \frac{I}{Z_0 + I (z - c t)} + I k \right)$

M.A.Sc. Thesis

Isaac Jing Herng Leung

06/07/96
Due to the symmetry of the sources, we can save some computation time by simply doubling our result. The top half of the array will be the same as the bottom half. The only exception being the line of sources at y = 0, for which they will not be doubled.

\[
fr := (x, y, z, t) \mapsto \sum_{i=-5}^{5} \text{rect} \left( -1.4 \times 10^{-6}, 7 \times 10^{-6}, t \right) \frac{\sqrt{\left( \Delta x \right)^2 + y^2 + z^2}}{c} \\Delta X \Delta Y \left/ \pi \right. 
\]

\[
\text{sqrt} \left( \left( x - \Delta x \right)^2 + y^2 + z^2 \right) \right) + \sum_{i=-5}^{5} \sum_{j=1}^{5} \frac{1}{2} \text{source} \left( \Delta x, \Delta y, 0, t \right) \frac{\sqrt{\left( x - \Delta x \right)^2 + \left( y - \Delta y \right)^2 + z^2}}{c} \\Delta X \Delta Y \left/ \pi \right. 
\]

\[
\text{rect} \left( -1.4 \times 10^{-5}, 7 \times 10^{-6}, t \right) \frac{\sqrt{\left( x - \Delta x \right)^2 + \left( y - \Delta y \right)^2 + z^2}}{c} \\Delta X \Delta Y \left/ \pi \right. 
\]

\[
\text{sqrt} \left( \left( x - \Delta x \right)^2 + \left( y - \Delta y \right)^2 + z^2 \right) \right) \right).
\]

> plot3d(Re(psi_FWM(z + 0.25/7, rho, 0.25/7/c)), z = -0.005/7..0.005/7, rho = 0..0.1/7, style = patch, axes = boxed, title = 'Theoretical FWM', labels = ['z (m)', 'radius out (m)', 'Re{phi}'], orientation = [-45, 45]);

> plot3d(Re(fr(x, 0, z + 0.25/7, 0.25/7/c)), z = -0.005/7..0.005/7, x = 0..0.1/7, style = patch, axes = boxed, title = 'Reconstructed FWM, 11x11 source, d = 25/7cm, delta = 0.00046');

> plot(Re(source(10*Delta[X], 0, 0, t)), t = -2E-6/3.5..2E-6/3.5, title = '3.5MHz, spacing = 0.46mm, location [10, 0]');
B.1.3 Matlab Program Files

This section contains the Matlab program files that were used to obtain the results presented previously. The program files were run under Matlab 4.2.

draw_fwm.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DRAW_FWM.M
% Date: March 27, 1996
% Author: Isaac J.H. Leung
% Description:
% This Matlab file is a helper program to calculate and plot
% a 3-D representation of a Focused Wave Mode (FWM) pulse on the
% current figure.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% N is the number of points to calculate, along the propagation axis
N=256;

% pN is the number of points to calculate, along the radius out from
% the propagation axis
pN=32;

% Speed of sound in water
c=1540;

% FWM Source spacing (in meters)
spacing=0.00092/2;

% Various other parameters for a FWM pulse
z0=0.6E-4;
k=12;

% d is the distance from the source along the propagation axis
% radius_out is the distance to measure out from the propagation axis.
% All distances in meters.
d=0.25/7;
radius_out=0.1/7;

% Here, calculate the theoretical FWM pulse at a given TIME out from
% the central source.
phi=zeros(pN,N);
t=0.25/7/c;
z=(-0.005/7+d):(2*0.005/7)/(N-1):(0.005/7+d);
rho=0:radius_out/(pN-1):radius_out;
for j=1:1:pN
phi(j,:)=1./(4*pi*i*(zO+i*(z-c*t))).*exp(i*k*(z+c*t)).*exp(-
k*rho(j)^2./(zO+i*(z-c*t)));

end

% Here, specify and calculate the specific FWM source pulse
% Note that this is _not_ required to calculate the actual FWM
% but is only required for the theoretical reconstruction.
% The source is NOT automatically plotted.
X=10*spacing;
Y=0*spacing;
tt=(-2E-6/3.5):2*(2E-6/3.5)/(N-1):(2E-6/3.5);
source = 1./(4*pi*i*(zO+i*(-c*tt))).*exp(i*k*(+c*tt)).*exp(-
k*(X^2+Y^2)./(zO+i*(-c*tt))).*( k-1./(zO+i*(-c*tt))+k*(X^2+Y^2)./(zO+i*(-
c*tt))).^2 ) ;

% Now plot the 3-D FWM pulse
%mesh(z,rho,real(phi));
surf(z,rho,real(phi));
%surf(z,rho,real(gwave));
shading interp;
colormap(bone);
view([30 30]);
grid;
xlabel('z (m)');
ylabel('rho (m)');
zlabel('Re{phi}');
axis([-0.005/7+d) (0.005/7+d) 0 radius_out -200 1200])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This section calculates the average frequency of the transmitted
% signal (freqin) and for the received signal (freqout)
for j=1:1:128
    ty=ty+abs(FSigOUT(j));
    tx=tx+abs(FSigIN(j));
    xx=xx+abs(FSigIN(j))*SigFreq(j);
    xy=xy+abs(FSigOUT(j))*SigFreq(j);
end

freqin=xx/tx
freqout=xy/ty

% Calculate the wavelength
lin=c/freqin;
lout=c/freqout;

% Calculate the near field for the transmitted signal (nfin) and
% also the received signal (nfout)
nfin=D^2/4/lin^2*(1-(lin/D)^2)
nfout=D^2/4/lout^2*(1-(lout/D)^2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f_source.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% F_SOURCE.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Date: April 19, 1996
% Author: Isaac J.H. Leung
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description:
% This Matlab function calculates the specified source signal
% required to generate a FWM pulse.
% Planar array is assumed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function src=make_source(x,y,z,t,k,z0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Various parameters for the FWM
% Note: We should externalize these parameters

c=1540;

src=i*(k*((x)^2+(y)^2)./((z0+i*(z-c*t)).^2)-1./(z0+i*(z-c*t))+k).*psi_FWM(0,sqrt(x.^2+y.^2),t,k,z0);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fwm_src.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

M.A.Sc. Thesis Isaac Jing Herng Leung 06/07/96
% FWM_SRC.M
%
% Date: April 19, 1996
% Author: Isaac J.H. Leung
%
% Description: This Matlab file calculates and plots the theoretical FWM pulse and also the reconstructed FWM pulse.

% c is the speed of sound in water
c=1540;

% N = number of points along z (propagation axis)
% rN = number of points to radially out from propagation axis
N=256;
rN=16;

% Source Element Spacing (in meters)
dx=0.0018;
dy=0.0018;
%dx=0.00092;
%dy=0.00092;

% Extent of Array Elements
xlim=10;
ylim=10;

% Parameters for the FWM
%z0=8E-4;
%k=88;
z0=4E-4; %z0=4E-4;
k=5;

fr=zeros(N,rN);
tfwm=zeros(N,rN);

% Set up the ranges to be considered
% Use the time to specify the distance out (t=distance/speed)
z=(128/20E6*c):(-256/20E6*c)/(N-1):(-128/20E6*c);
%z=8E-4:-16E-4/(N-1):-8E-4;
t=0.12/c;
x=0:(0.015)/(rN-1):0.015;

% Calculate the FWM pulses
for rr = 1:1:rN
    fr(:,rr)=real(fwmrecon(x(rr),0,z+t*c,t,k,z0,xlim,ylim,dx,dy,N));
    tfwm(:,rr)=real(psi_fwm(z+t*c,x(rr),t,k,z0));
end

% Plot the Theoretical Pulse
figure(1)
surf(x,z,real(tfwm/max(max(tfwm))));
%shading interp;
grid;
view([-45 45]);
colormap(jet);
xlabel('radius out (m)');
ylabel('z-ct (m)');
title('Theoretical FWM');

% Plot the (also Theoretical) but Reconstructed pulse
figure(2)
surf(x,z,fr/max(max(fr)));
%shading interp;
grid;
view([-45 45]);
colormap(jet);
xlabel('radius out (m)');
ylabel('z-ct (m)');
title('Reconstructed FWM');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fwmrecon.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FWMRECON.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Date: April 19, 1996
% Author: Isaac J.H. Leung
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function fr=fwmrecon(x,y,z,time,k,zO,xlim,ylim,dx,dy,N)
% c is the speed of sound in water
c=1540;
fr=zeros(N,1);
% These are the sources along y=0, of which there is only 1 set
for ii = -xlim:1:xlim
   tx=ii*dx;
   tt=(time-sqrt((x-tx)^2+(y-0)^2+(z).^2)/c);
   fr=fr+1*(-f_source(tx,0,0,tt,k,zO)*dx*dy/4/pi./sqrt((x-tx)^2+(y-0)^2+(z).^2));
end
% We only cycle through 1/2 the sources, since it's symmetrical
% we just multiply each result by 2.
for ii = -xlim:1:xlim
    for jj = 1:1:ylim
        tx=ii*dx;
        ty=jj*dy;
        tt=(time-sqrt((x-tx)^2+(y-ty)^2+(z)^2)/c);
        fr=fr+2*(-f_source(tx,ty,0,tt,k,z0)*dx*dy/4/pi./sqrt((x-tx)^2+(y-ty)^2+(z)^2));
    end % END jj LOOP
end % END ii LOOP

process.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROCESS.M
% Date: April 18, 1996
% Author: Isaac J.H. Leung
% Description:
% This Matlab file is a helper program to process received
% signals given a system transfer function.
% The system transfer function must be saved in the current directory
% as H.MAT
% Using this file, the program will read in the data file,
% assumed to be OUT.TXT and proceed to process the file using a
% Weiner filter and a low pass filter.
% The result is plotted on the current figure.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Specify the start and end points you want to analyze
pstart=20;
pend=pstart+255;

% Channel A is 1, Channel B is 2
channel=2;

% SN is the Number of points to be analyzed
SN=pend-pstart+1;

% ST is the time interval for the number of points
% Sampling Rate is default at 20 Msamples/sec.
% Sampling_Rate=20E6;
ST=SN/Sampling_Rate;
SigTime=0:ST/(SN-1):ST;

% Calculate the 'real' frequency for the axis
SigFreq = zeros(1,SN);
for j = 1:1:SN
    SigFreq(j) = j/ST;
end
% Load the Data files
% H.mat = System transfer Function
% out.txt = Received Signal
% fwm.txt = Calculated Signal
load H.mat
load fwm.txt
load out.txt

% Extract the data.
% More data is recorded than is needed, so you can choose to extract
% a specified range of data.
source=zeros(1,pend-pstart+1);
for j=1:1:size(fwm)
    source(j)=fwm(j)/max(fwm);
end

% Set y to be the portion of the data you're interested in.
% The mean of the signal is subtracted to remove any DC offset.
y = out(pstart:pend,channel)'-mean(out(:,channel));
Y = fft(y);

% Get a measure of the noise
if channel==2
    noise = 200E-3*out(pstart:pend,1)'
elseif channel==1
    noise = out(pstart:pend,2)'
end
NOISE = fft(noise);

% Constructing a Weiner filter, G(f)
Suu=Y.*conj(Y);
Snn=NOISE.*conj(NOISE);
G=(conj(H).*Suu)./((abs(H).^2).*Suu+l*Snn);

% Process Data
% X=Y./H;
X=Y.*G;
x=ifft(X);

% Making a Low Pass filter, LPF(f)
WN=hanning(SN);
LPF=zeros(1,SN);
for j=1:(SN/2);
    LPF(j)=WN(j+128);
end
for j=128:256
    LPF(j)=WN(j-127);
end
% XF=X.*LPF;
XF=X;
xf=(ifft(XF));
% Plot the Calculated Signal
subplot(3,1,1)
xi=zeros(size(SigTime));
for j=1:1:size(fwm);
    xi(j)=fwm(j)/max(fwm);
end
plot(SigTime,xi);
grid
axis([0 1.5E-5 -1 1])
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Calculated Signal');

% Plot the Received Signal
subplot(3,1,2)
plot(SigTime,y/max(y));
grid
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Received Signal');
axis([0 1.5E-5 -1 1])

% Plot the Reconstructed Signal
subplot(3,1,3)
plot(SigTime,real(xf)/abs(min(real(xf))));
grid
axis([0 1.5E-5 -1 1])
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Reconstructed Signal');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% PROCESSA.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Date: May 2, 1996
% Author: Isaac J.H. Leung
% Description:
% This Matlab file is a helper program to process received signals given a system transfer function.
% This procedure calculates the signal by integrating the received data. Correction for a varying DC offset is accomplished by curve fitting the offset.
% The result is plotted on the current figure.
% Specify the start and end points you want to analyze
pstart=20;

M.A.Sc. Thesis Isaac Jing Herng Leung 06/07/96
pend=pstart+255;

% Channel A is 1, Channel B is 2
channel=2;

% SN is the Number of points to be analyzed
SN=pend-pstart+1;

% ST is the time interval for the number of points
% Sampling Rate is default at 20 Msamples/sec.
Sampling_Rate=20E6;
ST=SN/Sampling_Rate;
SigTime=0:ST/(SN-1):ST;

% Calculate the 'real' frequency for the axis
SigFreq = zeros(1,SN);
for j = 1:1:SN
  SigFreq(j) = j/ST;
end

% Load the Data files
% out.txt  = Received Signal
% fwm.txt  = Calculated Signal
load fwm.txt
load out.txt

% Extract the data.
% More data is recorded than is needed, so you can choose to extract
% a specified range of data.
source=zeros(1,pend-pstart+1);
for j=1:1:size(fwm)
  source(j)=fwm(j)/max(fwm);
end

% Set y to be the portion of the data you're interested in.
y = out(pstart:pend,channel)';

% Do the first integration
yi = cumsum(y);
fl = polyfit(SigTime,yi,12);
yii = cumsum(yi-polyval(f1,SigTime));

% Plot the Calculated Signal
subplot(3,1,1)
plot(SigTime,fwm/max(fwm));
grid
axis([0 1.5E-5 -1 1])
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Calculated Signal');

% Plot the Received Signal
subplot(3,1,2)
plot(SigTime,y/max(y));
grid
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Received Signal');
axis([0 1.5E-5 -1 1])

% Plot the Reconstructed Signal
subplot(3,1,3)
plot(SigTime,yii/max(yii));
grid
axis([0 1.5E-5 -1 1])
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Reconstructed Signal');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

processf.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% PROCESSF.M
%
% Date: May 9, 1996
% Author: Isaac J.H. Leung
%
% Description:
% This matlab file is a helper program to process received
% signals given a system transfer function.
% This procedure calculates the signal by integrating the
% received data. Correction for a varying DC offset is accomplished
% by high pass filtering (Bartlett window).
% The result is plotted on the current figure.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Specify the start and end points you want to analyze
pstart=20;
pend=pstart+255;

% Channel A is 1, Channel B is 2
channel=2;

% SN is the Number of points to be analyzed
SN=pend-pstart+1;

% ST is the time interval for the number of points
% Sampling Rate is default at 20 Msamples/sec.
Sampling_Rate=20E6;
ST=SN/Sampling_Rate;
SigTime=0:ST/(SN-1):ST;

% Calculate the 'real' frequency for the axis
SigFreq = zeros(1,SN);
for j = 1:1:SN
    SigFreq(j) = j/ST;
%% Load the Data files
%% out.txt  = Received Signal
%% fwm.txt  = Calculated Signal
load fwm.txt
load out.txt

%% Extract the data.
%% More data is recorded than is needed, so you can choose to extract
%% a specified range of data.
source=zeros(1,pend-pstart+1);
for j=1:1:size(fwm)
    source(j)=fwm(j)/max(fwm);
end

%% Set y to be the portion of the data you're interested in.
y = out(pstart:pend,channel)';

%% Build the filter
CutFreq=3E5;
Wn=CutFreq/(Sampling_Rate/2);
hpf=firl(SN,Wn,'high',bartlett(SN+1));
hpf=firl(256,3E5/10E6,'high',bartlett(257));

%% Use the filter
yf=conv(hpf,y-mean(y));
%% Extract the portion we want. Convolution gives a longer vector.
yfe=yf(SN/2+1:SN/2+SN);

%% Do the first integration
yi = cumsum(yfe);
yii = cumsum(yi-mean(yi));

%% Plot the Calculated Signal
subplot(3,1,1)
plot(SigTime,fwm/max(fwm));
grid
axis([0 1.5E-5 -1 1])
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Calculated Signal');

%% Plot the Received Signal
subplot(3,1,2)
plot(SigTime,y/max(y));
grid
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Received Signal');
axis([0 1.5E-5 -1 1])

%% Plot the Reconstructed Signal
subplot(3,1,3)
plot(SigTime,yii/max(yii));
```matlab
grid
axis([0 1.5E-5 -1 1])
xlabel('time (s) --->');
ylabel('Amplitude --->');
title('Reconstructed Signal');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

psi_fwm.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PSI_FWM.M
%
% Date: April 19, 1996
% Author: Isaac J.H. Leung
%
% Description:
% This matlab function calculates the ideal theoretical FWM pulse (infinite, continuous source) at some distance z, at a given time t and at some rho out from the axis of propagation (z-axis).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function s=make_fwm(z,rho,t,k,zO)
% Various parameters for the FWM
% Note: We should externalize these parameters
c=1540;

s = 1/(4*pi*i) ./ (zO + i*(z-c*t)) .* exp(i*k*(z+c*t)) .* exp(-k*(rho^2) ./ (zO + i*(z-c*t)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

response.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% RESPONSE.M
%
% Date: March 27, 1996
% Author: Isaac J.H. Leung
%
% Description:
% This matlab file is a helper program to calculate the system response, stored in the variable H
% Using this file, the program will read in the data files:
% ROUT.TXT is assumed to be the received signal
% SINC.TXT is assumed to be the input signal
% this can be changed to whatever you choose the input signal to be.
% The characteristics of the input and output signal as well as the calculated system transfer function are plotted.

M.A.Sc. Thesis  Isaac Jing Herng Leung  06/07/96
% Assume that the received signal actually collects more data than needed.
% Specify the start and end points you want to analyze
pstart=55;
pend=pstart+255;

% SN is the Number of points to be analyzed
SN=pend-pstart+1;

% ST is the time interval for the number of points
% The default sampling rate is 20E6 samples/second.
Sampling_Rate=20E6;
ST=SN/Sampling_Rate;

SigIN=zeros(1,SN);
SigOUT=zeros(1,SN);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
% Here load the data files
load rout.txt;
load sinc.txt;

% Following this section, the received signal is always referred to
% as 'yout' and the calculated signal as 'in'
yout=rout;
in=sinc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
% Extracting the portion of the signal that we want.
DC_OFFSET=mean(yout(:,2));
for j=pstart:1:pend
  SigOUT(j-pstart+1)=yout(j,2)-DC_OFFSET;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%

for j=1:1:size(in)
  SigIN(j)=in(j);
end

% Fourier transform to get frequency spectrum.
FSigIN=fft(SigIN);
FSigOUT=fft(SigOUT);

% Calculate the 'real' time for the axis
SigTime=0:ST/(SN-1):ST;

% Calculate the 'real' frequency for the axis
SigFreq = zeros(1,SN);
for j = 1:1:SN
  SigFreq(j) = j/ST;
end

% Plot the INPUT signal
figure(1)
subplot(2,2,1)
plot(SigTime,SigIN,'g')
xlabel('seconds ---> ') ;
ylabel('Signal Amplitude (V) --->');
title('Original Source Signal');
grid on
axis([0 1.5E-5 -1 1])

subplot(2,2,3)
plot(SigFreq,abs(FSigIN),'c')
xlabel('Hz ---> ') ;
title('Signal Frequency Spectrum');
grid on
axis([0 10E6 0 5])

% Plot the OUTPUT signal
subplot(2,2,2)
plot(SigTime,SigOUT,'g')
xlabel('seconds ---> ') ;
ylabel('Signal Amplitude (V) --->');
title('Received Signal');
grid on
axis([0 1.5E-5 -1.0 1.0])

subplot(2,2,4)
plot(SigFreq,abs(FSigOUT),'c')
xlabel('Hz ---> ') ;
title('Signal Frequency Spectrum');
grid on
axis([0 10E6 0 5])

% Calculate the Transfer Function
H = zeros(1,SN);
for j = 1:1:SN
H(j) = FSigOUT(j)/FSigIN(j);
end

% Plot the Transfer Function
figure(2)
subplot(2,1,1)
semilogx(SigFreq, unwrap(angle(H))/2/pi*360, 'y');
xlabel('Hz ---> ') ;
ylabel('Phase (deg) --->');
title('System Phase Response');
grid on
axis([5E4 10E6 0 360])

subplot(2,1,2)
loglog(SigFreq,abs(H), 'm');
title('System Amplitude Response');
xlabel('Hz ---> ') ;
grid on
axis([5E4 10E6 1E-2 1E1])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
B.1.4 QuickBASIC Program Files

Programs for handling the Keithley Metrabyte M-STEP 3 Step Motor Controller were written using Microsoft QuickBASIC 4.0. This used because the supplied driver files are for QuickBASIC only. Drivers for other languages are available, but at extra cost. The required QuickBASIC driver file is called MSTEP.LIB

**make.bat**

echo This is the required command to actually compile the program.
c:\qb4\bin\bc f:\project\stepper\move6.bas /E/X/O
c:\qb4\bin\link /EX
MOVE6,f:\project\stepper\move6.exe,NUL,f:\project\stepper\mstep.lib;

**motion.cmd**

3 3938 40

**move6.bas**

```
DIM L$(17), P$(14), U$(14), D%(14), U(14), D$(3)
COMMON SHARED D%()
COMMON SHARED U()

DECLARE SUB QBMSTEP (MD%, BYVAL DUMMY%, STP&, FLAG%)
DECLARE SUB EmergencyStop (D%(), STP&, FLAG%)
DECLARE SUB ErrorHandler (FLAG%)
DECLARE SUB SelectChannel (D%())
DECLARE SUB InitializeParameters (D%(), STP&, FLAG%, U())
DECLARE SUB DisableInterrupt (D%(), STP&, FLAG%, U())
DECLARE SUB MoveRelative (D%(), STP&, FLAG%, Speed%)
```
DECLARE SUB MoveLimit (D%, STP&, FLAG%, Speed%)
DECLARE SUB MoveHome (D%, STP&, FLAG%, Speed%)
DECLARE SUB ReadStatus (D%, STP&, FLAG%, Speed%)

CLS
PRINT "****************
PRINT "* Move It V0.6 *
PRINT "****************

' Default parameters for initializing controllers
P$(1) = "Start rate divider: " : U(1) = 125 'slowest
P$(2) = "High speed run rate divider: " : U(2) = 20 'medium
P$(3) = "Acceleration/deceleration steps: " : U(3) = 200 '200 steps
P$(4) = "Motor type: " : U(4) = 2 '4 phase
P$(5) = "Excitation: " : U(5) = 1 'half step
P$(6) = "Logic polarity: " : U(6) = 1 'inverted
P$(7) = "Clock source: " : U(7) = 0 'internal
P$(8) = "Switching at standstill: " : U(8) = 1 'on
P$(9) = "ON time for power switch freq: " : U(9) = 16 'time on
P$(10) = "OFF time for power switch freq: " : U(10) = 32 'time off
P$(11) = "Port A direction, 0=input, 1=output: " : U(11) = 1 'output
P$(12) = "Port B direction, 0=input, 1=output: " : U(12) = 1 'output
P$(13) = "Auxiliary bit, 1=high, 0=low: " : U(13) = 0 'low
P$(14) = "Base I/O address: " : U(14) = &H310

' Declare other CALL variables
FLAG% = 0 'call error flag variable
STP& = 0 'step count (must be double precision)

REM Command% = 0
REM Distance& = 4000
Slow% = 200
Medium% = 100
Fast% = 20
REM Direction& = 1

CALL SelectChannel(D%)
CALL InitializeParameters (D%, STP&, FLAG%, U())

OPEN "MOTION.CMD" FOR INPUT AS #1
INPUT #1, Command%
SELECT CASE Command%
CASE IS > 7
    PRINT "Invalid Command"
CASE 6
    'What's the status?
    CALL ReadStatus (D%, STP&, FLAG%)
CASE 5
'Move it home
INPUT #1, Direction%
INPUT #1, StepSpeed%
CALL MoveHome(D%(()), Direction%, FLAG%, StepSpeed%)

CASE 4
'Move it to the limits
INPUT #1, Direction%
INPUT #1, StepSpeed%
CALL MoveLimit(D%(()), Direction%, FLAG%, StepSpeed%)

CASE 3
'Move the darn thing a bit
'Note that it currently uses acceleration/deceleration
'and that in this mode, SPEED isn't used. It's always high
INPUT #1, Distance%
INPUT #1, StepSpeed%
CALL MoveRelative(D%(()), Distance%, FLAG%, StepSpeed%)

CASE 2
'Turn off the Interrupts
CALL DisableInterrupt(D%(()), STP&, FLAG%)

CASE 1
'Initialize all parameters
IF Command% = 1 THEN CALL InitializeParameters(D%(()), STP&,
FLAG%, U())

CASE 0
'Initialize Channel A
CALL SelectChannel(D%(()))

CASE -1
'Emergency Stop
CALL EmergencyStop(D%(()), STP&, FLAG%)

END SELECT

CLOSE #1

END

SUB DisableInterrupt (D%(()), STP&, FLAG%)

'--- MODE 14: Enable/disable interrupt -----------------------------

REM * We're going to just disable to interrupt.
MD% = 14

INTLEV% = 5
D%(2) = 0
D%(3) = INTLEV%
PRINT "MSTEP-3 => Disable Interrupt"
PRINT " Interrupt Level = "; INTLEV%
CALL QBMSTEP(MD%, VARPTR(D%(0)), STP&, FLAG%)
IF FLAG% <> 0 THEN CALL ErrorHandler(FLAG%)
END SUB

SUB EmergencyStop (D%(), STP&, FLAG%)

' --- MODE 0: Emergency stop --------------------------------------------
MD% = 0
PRINT "MSTEP-3 => Emergency Stop"
CALL QBMSTEP(MD%, VARPTR(D%(0)), STP&, FLAG%)
IF FLAG% <> 0 THEN CALL ErrorHandler(FLAG%)
END SUB

SUB ErrorHandler (FLAG%)

4120 ' --- Error code routine ---------------------------------------------
PRINT "MSTEP-3 => Error Handler"
PRINT : PRINT "Driver has returned error code "; FLAG%
PRINT : PRINT "This was caused by: -": PRINT
IF FLAG% = 1 THEN PRINT " Motor busy executing a command"
IF FLAG% = 2 THEN PRINT " Driver not initialized on Channel A"
IF FLAG% = 3 THEN PRINT " Driver not initialized on Channel B"
IF FLAG% = 4 THEN PRINT " Driver not initialized on Channel C"
IF FLAG% = 5 THEN PRINT " Mode number <0 or >12"
IF FLAG% = 6 THEN PRINT " A hardware error e.g. Wrong I/O address, PPMC controller failure"
IF FLAG% = 7 THEN PRINT " Step count <-16,777,215 or >+16,777,215"
IF FLAG% = 8 THEN PRINT " Motor already at standstill. Command cannot be executed"
IF FLAG% >= 10 AND FLAG% <= 24 THEN PRINT " Error in range of data variable D%("; FLAG% - 10; ")"
PRINT
INPUT "Press <ENTER> to continue... ", DUMMY$
END SUB

SUB InitializeParameters (D%(), STP&, FLAG%, U())

' --- MODE 15: Initialize -----------------------------------------------
MD% = 15
PRINT "MSTEP-3 => Initialize controller card"
PRINT " Base I/O Address = 784"
PRINT " Half Step Mode"
FOR I% = 1 TO 14
IF U(I%) > 65535! OR U(I%) < -32768! THEN D%(I%) = -1: GOTO 3550
IF U(I%) > 32767 THEN D%(I%) = U(I%) - 65536! ELSE D%(I%) = U(I%)
3550 NEXT I%

M.A.Sc. Thesis Isaac Jing Herng Leung 06/07/96
CALL QBMSTEP(MD%, VARPTR(D%(0)), STP&, FLAG%)
REM * This is Just error checking, I think.
IF FLAG% <> 0 THEN CALL ErrorHandler(FLAG%)
END SUB

SUB MoveHome (D%(), STP&, FLAG%, Speed%)
    '--- MODE 7: Constant speed to home point -----------------------------
    MD% = 7
    PRINT "MSTEP-3 => Move to home point"  
    PRINT " (No acceleration)"
    D%(1) = Speed%
    CALL QBMSTEP(MD%, VARPTR(D%(0)), STP&, FLAG%)
    IF FLAG% <> 0 THEN CALL ErrorHandler(FLAG%)
END SUB

SUB MoveLimit (D%(), STP&, FLAG%, Speed%)
    '--- MODE 5: Constant speed to limit -----------------------------
    MD% = 5
    PRINT "MSTEP-3 => Move to limit"  
    PRINT " (No acceleration)"
    D%(1) = Speed%
    CALL QBMSTEP(MD%, VARPTR(D%(0)), STP&, FLAG%)
    IF FLAG% <> 0 THEN CALL ErrorHandler(FLAG%)
END SUB

SUB MoveRelative (D%(), STP&, FLAG%, Speed%)
    '--- MODE 4: Constant speed -----------------------------
    'MD% = 4
    '--- MODE 3: Accelerate/Decelerate -----------------------------
    MD% = 3
    PRINT "MSTEP-3 => Move Relative"
    PRINT " Distance = "; STP&
    PRINT " Speed = "; Speed%
    D%(1) = Speed%
1338 CALL QBMSTEP(MD%, VARPTR(D%(0)), STP&, FLAG%)
    IF FLAG% <> 0 THEN CALL ErrorHandler(FLAG%)
END SUB

SUB ReadStatus (D%(), STP&, FLAG%)
    '--- MODE 8: Read status -----------------------------
    MD% = 8
PRINT "MSTEP-3 => Read Status"
PRINT ""
CALL QBMSTEP(MD%, VARPTR(D%(0)), STP&, FLAG%)
IF FLAG% <> 0 THEN CALL ErrorHandler(FLAG%)

PRINT "STATUS DATA RETURNED:"
PRINT " Finish status = "; HEX$(D%(2)); " hex"
PRINT " Input status = "; HEX$(D%(3)); " hex"
PRINT " Output status = "; HEX$(D%(4)); " hex"
PRINT " Remaining steps = "; STP&
PRINT "From this status data, the following individual data can be extracted:"
PRINT "From FINISH STATUS"
PRINT "Last motion command was mode "; (D%(2) AND &H7)
IF (D%(2) AND &H8) = &H8 THEN PRINT "Motor performed decelerating stop at high speed limit L3 or L4"
IF (D%(2) AND &H10) = &H10 THEN PRINT "Motor performed decelerating stop at limit L1 or L2"
IF (D%(2) AND &H20) = &H20 THEN PRINT "Motor enable control MC is disabled"
IF (D%(2) AND &H40) = &H40 THEN PRINT "Motor was stopped by a stop command and did not complete pulse count"
IF (D%(2) AND &H78) = 0 THEN PRINT "Command completed normally" ELSE PRINT "Command aborted before travelling set number of steps"
IF (D%(2) AND &H80) = &H80 THEN PRINT "Last command interrupt bit set" ELSE PRINT "Last command interrupt bit cleared"
PRINT : PRINT "From INPUT STATUS"
IF (D%(3) AND &H1) = 0 THEN PRINT "Limit input L4 asserted"
IF (D%(3) AND &H2) = 0 THEN PRINT "Limit input L3 asserted"
IF (D%(3) AND &H4) = 0 THEN PRINT "Limit input L2 asserted"
IF (D%(3) AND &H8) = 0 THEN PRINT "Limit input L1 asserted"
IF (D%(3) AND &H40) = 0 THEN PRINT "Motor on signal MC is disabling controller"
IF (D%(3) AND &H80) = 0 THEN PRINT "Motor is at base point reference"
IF (D%(3) AND &HCF) = &HCF- THEN PRINT "All limit inputs are high (not asserted)"
PRINT : PRINT "From OUTPUT STATUS"
PRINT "Phase outputs S1-S5 are at logic levels ";
FOR I% = 7 TO 3 STEP -1
IF (D%(4) AND 2 ^ I%) = 2 ^ I% THEN PRINT "1 " ELSE PRINT "0 ";
NEXT I%
PRINT "Press <ENTER> to continue...", DUMMY&

END SUB

SUB SelectChannel (D%())

'--- Pseudo mode 16: Get channel A, B or C on MD%=16 and place in D%(0) ---'
PRINT "MSTEP-3 => Select Channel A"

' This sets the Channel to A
D%(0) = 0

END SUB